

**Defect Management Process** Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

**1. What is A Defect?** A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

**2. Defect Management Life Cycle** When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

**3. Defect Life Cycle in Detail** The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

**3.1. Defect Workflow** The diagram below shows the actual life cycle of a defect.

**3.2. Defect States**

- #1) **New:** This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) **Assigned:** In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) **Open:** Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

a) **Rejected:** If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.

b) **Duplicate:** If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.

c) **Deferred:** If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in

such a case, developer can change the status of the defect as 'Deferred'. d) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug". #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed". #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest". #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 3.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 4. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 5. Defect Management

#### 5.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

##### 5.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

##### 5.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

#### 5.2. Testing stages and defects

Responsibilities SL No. Testing Stage Environment Defect Owner Defect Retested By Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team 4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager

### 5.3. Defect Triage Calls

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

5.3.1.Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

5.3.2.How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

5.4.Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

5.5.Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

5.5.1.Defect Trends by Release

- o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

5.5.2.Defects Density

- o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.
- o Defect Density = Total Defect/Size of the Release

5.5.3.Defect Fix Rate per Release

- o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .
- o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

5.5.4.Defect Aging

- o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect

was fixed (if the defect is already fixed). 5.5.5.Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization. o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms. Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish. Follow the steps below to create a fishbone diagram: (1) Write the problem at the head of the fish. (2) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N] (3) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N. (4) Extend the causes to secondary, tertiary, and more levels as applicable. o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

### 5.6.Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

#### 5.6.1.Action items, Action Owners, Tracking, Checklist etc.

6. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was resolved and recommendations for testing.
- References

#### 7. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does

an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted 3 A minor incident with low impact • A minor inconvenience to customers, workaround available In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs Blocker — Highest priority. Indicates that this issue takes precedence over all others. Critical — Indicates that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority.

8. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 9. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

10. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

11. Defect Management Life Cycle

When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or

rejected by a development team. The number of states that a defect goes through varies from project to project.

### 12. Defect Life Cycle in Detail

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

#### 12.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

#### 12.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.
- e) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- f) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.
- g) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.
- h) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".
- #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".
- #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".
- #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.
- #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.
- #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.
- #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 12.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 13. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 14. Defect Management

#### 14.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and

business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

14.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

14.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

14.2. Testing stages and defects Responsibilities SL No. Testing Stage Environment Defect Owner Defect Retested By Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team 4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager 14.3. Defect Triage Calls

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

14.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

14.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

14.4. Defect Metrics Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA,

UAT or End-user 14.5. Defect Reporting Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

14.5.1. Defect Trends by Release o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

14.5.2. Defects Density o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

14.5.3. Defect Fix Rate per Release o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

14.5.4. Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

14.5.5. Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish. Follow the steps below to create a fishbone diagram:

- (5) Write the problem at the head of the fish.
- (6) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
- (7) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
- (8) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

14.6. Defect Prevention Processes & Improvements Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

#1) Review and Inspection: This

method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products. #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system. #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects. #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

#### 14.6.1. Action items, Action Owners, Tracking, Checklist etc.

15. Defect Data • Name of the Person • Types of Testing • Problem Summary • Detailed Description of Defect. • Steps to Reproduce • Life Cycle Phase • Work product where Defect was introduced. • Severity and Priority • Subsystem or Component where the Defect is introduced. • Project Activity occurs when the Defect is introduced. • Identification Method • Type of Defect • Projects and Products in which problems exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References

#### 16. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples

- 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss
- 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted
- 3 A minor incident with low impact • A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

#### 17. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 18. Conclusion

This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

#### 19. What is A Defect?

A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

#### 20. Defect Management Life Cycle

When a system gives a different output other than the actual

business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

### 21. Defect Life Cycle in Detail

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

#### 21.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

#### 21.2. Defect States #1)

New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.

#2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.

#3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

- i) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- j) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.
- k) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.
- l) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".

#4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".

#5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".

#6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.

#7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.

#8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.

#9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields

where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 21.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 22. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 23. Defect Management

#### 23.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

#### 23.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

#### 23.1.2. Why is Defect Tracking Necessary?

- A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- Monitors the bugs that had already been resolved, thereby saving time.
- To ensure that right defects are being worked on.

#### 23.2. Testing stages and defects Responsibilities

SL No. Testing Stage Environment Defect Owner Defect Retested By

Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team

4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager

#### 23.3. Defect Triage

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

#### 23.3.1. Thumb rules of Triaging

- All reported defects have been reviewed
- All accepted defects have been prioritized
- All accepted defects have severity attached
- All rejected defects should have reasonable explanations for the testing team
- Every defect has been assigned to an appropriate owner, individual or team
- Root cause analysis for every accepted defect has been done

#### 23.3.2. How does Defect Triaging work?

The

defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

### 23.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

### 23.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

#### 23.5.1. Defect Trends by Release

o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

#### 23.5.2. Defects Density

o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

#### 23.5.3. Defect Fix Rate per Release

o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

#### 23.5.4. Defect Aging

o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

#### 23.5.5. Defect Analysis through RCA

o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
- Follow the steps below to create a fishbone diagram:
  - (9) Write the problem at the head of the fish.
  - (10) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
  - (11) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
  - (12) Extend the causes to

secondary, tertiary, and more levels as applicable.

- o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

23.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

23.6.1. Action items, Action Owners, Tracking, Checklist etc.

24. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was resolved and recommendations for testing.
- References

25. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples

- 1 A critical incident with very high impact
- A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss
- 2 A major incident with significant impact
- A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted
- 3 A minor incident with low impact
- A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial

- Bugs
- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

26. Defect Fix timeline

Severity Response Time Resolution Time

- 1
- 2
- 3
- 4
27. Conclusion

This was all there is to know about the Bug Life Cycle and how the testing

and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

28. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

29. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

30. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

30.1. Defect Workflow The diagram below shows the actual life cycle of a defect.

30.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

m)

Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer. n) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer. o) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'. p) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug". #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed". #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest". #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

30.3. Guidelines for Implementing a Defect Life Cycle Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

31. Goals of Defect Management Process (DMP) Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

32. Defect Management

32.1. Defect Tracking In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

32.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

32.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby

saving time. o To ensure that right defects are being worked on.

### 32.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Defect Retested By
1	Unit Testing (UT)	DEV	DEV Team	Dev Team
2	System Testing (ST)	ST	ST Team	ST Team
3	System Integration Testing (SIT)	SIT	SIT Team	SIT Team
4	User Acceptance Testing (UAT)	UAT	UAT Team	UAT Team

### 32.3. Defect Triage

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

#### 32.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

#### 32.3.2. How does Defect Triage work?

The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

### 32.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

### 32.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

#### 32.5.1. Defect Trends by Release

The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

#### 32.5.2. Defects Density

o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

$$\text{o Defect Density} = \frac{\text{Total Defect}}{\text{Size of the software}}$$

Release 32.5.3. Defect Fix Rate per Release o Defect removal efficiency = (1 - (total defects caught by customer/ total number of defects)) x 100. o Residual defect density = (total number of defects found by a customer)/ (Total number of defects including customer found defects) x 100. 32.5.4. Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed). 32.5.5. Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization. o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms. Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish. Follow the steps below to create a fishbone diagram: (13)Write the problem at the head of the fish. (14)Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N] (15)Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N. (16)Extend the causes to secondary, tertiary, and more levels as applicable. o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

32.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

32.6.1. Action items, Action Owners, Tracking, Checklist etc.

33. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.

**Impact on Project** • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References 34. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted 3 A minor incident with low impact • A minor inconvenience to customers, workaround available In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs Blocker — Highest priority. Indicates that this issue takes precedence over all others. Critical — Indicates that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority. 35. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 36. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process. 37. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 38. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In

order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

### 39. Defect Life Cycle in Detail

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

#### 39.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

#### 39.2. Defect States

#1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.

#2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.

#3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

q) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.

r) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.

s) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.

t) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".

#4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".

#5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".

#6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.

#7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.

#8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.

#9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 39.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 40. Goals of Defect Management Process

(DMP) Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

41. Defect Management 41.1. Defect Tracking In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

41.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

41.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

41.2. Testing stages and defects Responsibilities SL No. Testing Stage Environment Defect Owner Defect Retested By Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team 4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager 41.3. Defect Triage Calls Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

41.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

41.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

41.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

41.5. Defect Reporting Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

41.5.1. Defect Trends by Release o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

41.5.2. Defects Density o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

41.5.3. Defect Fix Rate per Release o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

41.5.4. Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

41.5.5. Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish. Follow the steps below to create a fishbone diagram:

- (17) Write the problem at the head of the fish.
- (18) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
- (19) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
- (20) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
- 4.

Group Problems Together 5. Add up Scores for Each Group 6. Take Action 41.6. Defect Prevention Processes & Improvements Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks: #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products. #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system. #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects. #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis 41.6.1. Action items, Action Owners, Tracking, Checklist etc. 42. Defect Data • Name of the Person • Types of Testing • Problem Summary • Detailed Description of Defect. • Steps to Reproduce • Life Cycle Phase • Work product where Defect was introduced. • Severity and Priority • Subsystem or Component where the Defect is introduced. • Project Activity occurs when the Defect is introduced. • Identification Method • Type of Defect • Projects and Products in which problems exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References 43. Defect Priority and Severity Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted 3 A minor incident with low impact • A minor inconvenience to customers, workaround available In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs Blocker — Highest priority. Indicates that this issue takes precedence over all others. Critical — Indicates that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority. 44. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 45. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process. 46. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The

defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 47.

**Defect Management Life Cycle** When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project. 48. Defect Life Cycle in Detail

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

48.1. Defect Workflow The diagram below shows the actual life cycle of a defect. 48.2. Defect States #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle. #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer. #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below

namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason. u) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as

"Rejected" by the developer. v) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer. w) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'. x) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug". #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".

#5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest". #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the

requirements or not. #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

48.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

49. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

50. Defect Management

50.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

50.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

50.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

50.2. Testing stages and defects Responsibilities

SL No. Testing Stage Environment Defect Owner Defect Retested By Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team 4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager

50.3. Defect Triage

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should

have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

50.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team

Every defect has been assigned to an appropriate owner, individual or team

- o Root cause analysis for every accepted defect has been done

50.3.2. How does Defect Triaging work?

The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

50.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

50.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

50.5.1. Defect Trends by Release

- o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

50.5.2. Defects Density

- o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.
- o Defect Density = Total Defect/Size of the Release

50.5.3. Defect Fix Rate per Release

- o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100.$
- o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100.$

50.5.4. Defect Aging

- o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

50.5.5. Defect Analysis through RCA

- o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.
- o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its

symptoms. Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish. Follow the steps below to create a fishbone diagram: (21)Write the problem at the head of the fish. (22)Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N] (23)Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N. (24)Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

#### 50.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

#### 50.6.1. Action items, Action Owners, Tracking, Checklist etc.

51. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was resolved and recommendations for testing.
- References

52. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first?

Severity Description Examples

- 1 A critical incident with very high impact
- A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss
- 2 A major incident with significant impact
- A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted
- 3 A minor incident with low impact
- A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs Blocker — Highest priority. Indicates that this issue takes precedence over all others. Critical — Indicates that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority.

53. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 5 4. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

55. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

56. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

57. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

57.1. Defect Workflow The diagram below shows the actual life cycle of a defect.

57.2. Defect States #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this

defect in the later stages of the Defect Life Cycle. #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer. #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason. y) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer. z) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer. aa) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'. bb) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug". #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed". #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest". #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

57.3. Guidelines for Implementing a Defect Life Cycle Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

58. Goals of Defect Management Process (DMP) Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

59. Defect Management 59.1. Defect Tracking In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

59.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all

the defects. So that the management may not miss any defect from correction. o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects. o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

59.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

59.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Defect Retested By
1	Unit Testing (UT)	DEV	DEV Team	Dev Team
2	System Testing (ST)	ST	ST	Team
3	System Integration Testing (SIT)	SIT	SIT	Team
4	User Acceptance Testing (UAT)	UAT	UAT Team	UAT Team
				Test Manager

59.3. Defect Triage

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

59.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

59.3.2. How does Defect Triaging work?

The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

59.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

59.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect

situation to get feedback from them.

59.5.1. Defect Trends by Release o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

59.5.2. Defects Density o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

59.5.3. Defect Fix Rate per Release o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

59.5.4. Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

59.5.5. Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish. Follow the steps below to create a fishbone diagram:

- (25) Write the problem at the head of the fish.
- (26) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
- (27) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
- (28) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

59.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

59.6.1. Action items,

Action Owners, Tracking, Checklist etc. 60. Defect Data • Name of the Person • Types of Testing • Problem Summary • Detailed Description of Defect. • Steps to Reproduce • Life Cycle Phase • Work product where Defect was introduced. • Severity and Priority • Subsystem or Component where the Defect is introduced. • Project Activity occurs when the Defect is introduced. • Identification Method • Type of Defect • Projects and Products in which problems exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References 61. Defect Priority and Severity Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted 3 A minor incident with low impact • A minor inconvenience to customers, workaround available In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs Blocker — Highest priority. Indicates that this issue takes precedence over all others. Critical — Indicates that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority. 62. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 63. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process. 64. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 65. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but

along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

### 66. Defect Life Cycle in Detail

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

#### 66.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

#### 66.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.
- cc) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- dd) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.
- ee) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.
- ff) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".
- #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".
- #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".
- #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.
- #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.
- #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.
- #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 66.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her

responsibility very clearly for better results. • Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily. • The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

67. Goals of Defect Management Process (DMP) Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

68. Defect Management

68.1. Defect Tracking In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

68.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

68.1.2. Why is Defect Tracking Necessary?

- A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- Monitors the bugs that had already been resolved, thereby saving time.
- To ensure that right defects are being worked on.

68.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Defect Retested By	Defect Tracking By	1 Unit Testing (UT)	DEV DEV Team	Dev Team	2 System Testing (ST)	ST ST Team	ST Team	ST Team	ST Team	3 System Integration Testing (SIT)	SIT SIT Team	SIT Team	SIT Team	4 User Acceptance Testing (UAT)	UAT UAT Team	UAT Team	Test Manager
--------	---------------	-------------	--------------	--------------------	--------------------	---------------------	--------------	----------	-----------------------	------------	---------	---------	---------	------------------------------------	--------------	----------	----------	---------------------------------	--------------	----------	--------------

68.3. Defect Triage

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

68.3.1. Thumb rules of Triaging

- All reported defects have been reviewed
- All accepted defects have been prioritized
- All accepted defects have severity attached
- All rejected defects should have reasonable explanations for the testing team
- Every defect has been assigned to an appropriate owner, individual or team
- Root cause analysis for every accepted defect has been done

68.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time

sensitivity for resolution • The complexity involved in the resolution • Business impact This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

#### 68.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

#### 68.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

##### 68.5.1. Defect Trends by Release

o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

##### 68.5.2. Defects Density

o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

##### 68.5.3. Defect Fix Rate per Release

o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

##### 68.5.4. Defect Aging

o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

##### 68.5.5. Defect Analysis through RCA

o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
- Follow the steps below to create a fishbone diagram:

- (29) Write the problem at the head of the fish.
- (30) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
- (31) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
- (32) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement

Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit.

The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent

of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

### 6.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

#### 6.6.1. Action items, Action Owners, Tracking, Checklist etc.

6.9. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was resolved and recommendations for testing.
- References

#### 70. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples

- 1 A critical incident with very high impact
- A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss

- 2 A major incident with significant impact
- A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted

- 3 A minor incident with low impact
- A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

#### 71. Defect Fix timeline

Severity Response Time Resolution Time

1	2	3
4	72.	Conclusion

This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention

is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process. 73. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 74.

Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project. 75. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again. 75.1. Defect Workflow The diagram below shows the actual life cycle of a defect. 75.2. Defect States #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle. #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer. #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason. gg) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer. hh) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer. ii) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'. Defect Management Process Defect management process is mainly used to improve the efficiency of the software

development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

1. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

2. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

3. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

3.1. Defect Workflow The diagram below shows the actual life cycle of a defect.

3.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

a) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.

b) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.

c) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.

d) Not a Bug: If the

defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug". #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed". #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest". #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 3.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 4. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 5. Defect Management

#### 5.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

##### 5.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

##### 5.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

#### 5.2. Testing stages and defects Responsibilities

SL No. Testing Stage Environment Defect Owner Defect Retested By Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team 4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager

### 5.3. Defect Triage Calls

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence.

Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

5.3.1.Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

5.3.2.How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

5.4.Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

5.5.Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

5.5.1.Defect Trends by Release

- o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

5.5.2.Defects Density

- o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.
- o Defect Density = Total Defect/Size of the Release

5.5.3.Defect Fix Rate per Release

- o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .
- o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

5.5.4.Defect Aging

- o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

5.5.5.Defect Analysis through RCA

- o RCA (Root Cause

Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

- o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.

Follow the steps below to create a fishbone diagram:

- (1) Write the problem at the head of the fish.
- (2) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
- (3) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
- (4) Extend the causes to secondary, tertiary, and more levels as applicable.

- o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement

Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit.

The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes.

Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

### 5.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects.

Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

#### 5.6.1. Action items, Action Owners, Tracking, Checklist etc.

6. Defect Data
  - Name of the Person
  - Types of Testing
  - Problem Summary
  - Detailed Description of Defect.
  - Steps to Reproduce
  - Life Cycle Phase
  - Work product where Defect was introduced.
  - Severity and Priority
  - Subsystem or Component where the Defect is introduced.
  - Project Activity occurs when the Defect is introduced.
  - Identification Method
  - Type of Defect
  - Projects and Products in which problems exist
  - Current Owner
  - Current State of the Report
  - Work product where Defect occurred.
  - Impact on Project
  - Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
  - Dates when various defect lifecycle phases occur.
  - Description of how the defect was resolved and recommendations for testing.
  - References
7. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing

a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted 3 A minor incident with low impact • A minor inconvenience to customers, workaround available In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs Blocker — Highest priority. Indicates that this issue takes precedence over all others. Critical — Indicates that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority. 8. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 9. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process. 10. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 11. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from

project to project.

## 12. Defect Life Cycle in Detail

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

### 12.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

### 12.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.
- e) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- f) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.
- g) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.
- h) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".
- #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".
- #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".
- #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.
- #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.
- #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.
- #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 12.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

## 13. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

## 14. Defect Management

### 14.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing,

evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

14.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

14.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

14.2. Testing stages and defects Responsibilities SL No. Testing Stage Environment Defect Owner Defect Retested By Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team 4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager

14.3. Defect Triage Calls Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

14.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

14.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

14.4. Defect Metrics Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

14.5. Defect Reporting Defect Reporting in software testing is a process in

which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

14.5.1. Defect Trends by Release o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

14.5.2. Defects Density o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

14.5.3. Defect Fix Rate per Release o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

14.5.4. Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

14.5.5. Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish. Follow the steps below to create a fishbone diagram:

- (5) Write the problem at the head of the fish.
- (6) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
- (7) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
- (8) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

14.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and

inspection of all work products. #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system. #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects. #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

#### 14.6.1. Action items, Action Owners, Tracking, Checklist etc.

15. Defect Data • Name of the Person • Types of Testing • Problem Summary • Detailed Description of Defect. • Steps to Reproduce • Life Cycle Phase • Work product where Defect was introduced. • Severity and Priority • Subsystem or Component where the Defect is introduced. • Project Activity occurs when the Defect is introduced. • Identification Method • Type of Defect • Projects and Products in which problems exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References

#### 16. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples

- 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss
- 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted
- 3 A minor incident with low impact • A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

#### 17. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 18. Conclusion

This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

#### 19. What is A Defect?

A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

#### 20. Defect Management Life Cycle

When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business

requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

### 21. Defect Life Cycle in Detail

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

#### 21.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

#### 21.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.
- i) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- j) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.
- k) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.
- l) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".
- #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".
- #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".
- #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.
- #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.
- #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.
- #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase

and Reporter. 21.3. Guidelines for Implementing a Defect Life Cycle Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

22. Goals of Defect Management Process (DMP) Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

23. Defect Management 23.1. Defect Tracking In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

23.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

23.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

23.2. Testing stages and defects Responsibilities SL No. Testing Stage Environment Defect Owner Defect Retested By Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team

4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager 23.3. Defect Triage Calls Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

23.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

23.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes

stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

### 23.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

### 23.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

#### 23.5.1. Defect Trends by Release

The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

#### 23.5.2. Defects Density

o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

#### 23.5.3. Defect Fix Rate per Release

o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

#### 23.5.4. Defect Aging

o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

#### 23.5.5. Defect Analysis through RCA

o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
- Follow the steps below to create a fishbone diagram:
  - (9) Write the problem at the head of the fish.
  - (10) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
  - (11) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
  - (12) Extend the causes to secondary, tertiary, and more levels as applicable.
- o Parato charts to do the affinity analysis A

Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement. Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

### 23.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

#### 23.6.1. Action items, Action Owners, Tracking, Checklist etc.

24. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was resolved and recommendations for testing.
- References

25. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first?

Severity Description Examples

- 1 A critical incident with very high impact
- A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss

2 A major incident with significant impact

- A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted

3 A minor incident with low impact

- A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

26. Defect Fix timeline

Severity Response Time Resolution Time

1	2	3
4	27. Conclusion	This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-

free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

28. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

29. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

30. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

30.1. Defect Workflow The diagram below shows the actual life cycle of a defect.

30.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.
- m) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked

as "Rejected" by the developer. n) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer. o) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'. p) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug". #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed". #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest". #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 30.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 31. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 32. Defect Management

#### 32.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

#### 32.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

#### 32.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

### 32.2. Testing stages and

defects Responsibilities SL No. Testing Stage Environment Defect Owner Defect Retested By Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team 4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager 32.3. Defect Triage Calls Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test. 32.3.1. Thumb rules of Triaging o All reported defects have been reviewed o All accepted defects have been prioritized o All accepted defects have severity attached o All rejected defects should have reasonable explanations for the testing team o Every defect has been assigned to an appropriate owner, individual or team o Root cause analysis for every accepted defect has been done 32.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are: • The validity of the defect • Time sensitivity for resolution • The complexity involved in the resolution • Business impact This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced. 32.4. Defect Metrics Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be: • Defect distribution by cause • Defect distribution by feature/functional area • Defect distribution by Severity • Defect distribution by Priority • Defect distribution by type • Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user 32.5. Defect Reporting Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them. 32.5.1. Defect Trends by Release o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed. 32.5.2. Defects Density o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software. o Defect Density = Total Defect/Size of the Release 32.5.3. Defect Fix Rate per Release o Defect removal efficiency = (1 - (total defects

caught by customer/ total number of defects)) x 100. o Residual defect density = (total number of defects found by a customer)/ (Total number of defects including customer found defects) x 100.

32.5.4. Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

32.5.5. Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish. Follow the steps below to create a fishbone diagram:

- (13) Write the problem at the head of the fish.
- (14) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
- (15) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
- (16) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

32.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

32.6.1. Action items, Action Owners, Tracking, Checklist etc.

33. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the

defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References 34. Defect Priority and Severity Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted 3 A minor incident with low impact • A minor inconvenience to customers, workaround available In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs Blocker — Highest priority. Indicates that this issue takes precedence over all others. Critical — Indicates that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority. 35. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 36. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process. 37. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 38. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect

Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

### 39. Defect Life Cycle in Detail

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

#### 39.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

#### 39.2. Defect States

#1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.

#2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.

#3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

q) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.

r) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.

s) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.

t) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".

#4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".

#5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".

#6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.

#7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.

#8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.

#9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 39.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 40. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection

- Minimize the impact • Resolution of the Defect • Process improvement

**41. Defect Management**

**41.1. Defect Tracking** In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

**41.1.1. Objectives of Defect Tracking** A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

**41.1.2. Why is Defect Tracking Necessary?**

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

**41.2. Testing stages and defects Responsibilities**

SL No. Testing Stage Environment Defect Owner Defect Retested By

Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST

Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team

4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager

**41.3. Defect Triage** Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

**41.3.1. Thumb rules of Triaging**

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

**41.3.2. How does Defect Triaging work?** The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be

considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

**41.4. Defect Metrics** Software quality assurance metrics must be used to track defects and structure the process of

their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

41.5. Defect Reporting Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

41.5.1. Defect Trends by Release o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

41.5.2. Defects Density o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

41.5.3. Defect Fix Rate per Release o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

41.5.4. Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

41.5.5. Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish. Follow the steps below to create a fishbone diagram:

- (17) Write the problem at the head of the fish.
- (18) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
- (19) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
- (20) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

41.6. Defect

**Prevention Processes & Improvements** Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches:
  - I) Pareto Analysis and II) Fishbone Analysis

41.6.1. Action items, Action Owners, Tracking, Checklist etc.

42. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was resolved and recommendations for testing.
- References

43. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first?

Severity Description Examples

- 1 A critical incident with very high impact
- A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss
- 2 A major incident with significant impact
- A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted
- 3 A minor incident with low impact
- A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

44. Defect Fix timeline

Severity Response Time Resolution Time

1	2	3
---	---	---

45. Conclusion

This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

46. What is A Defect?

A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an

application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 47.

**Defect Management Life Cycle** When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project. 48.

**Defect Life Cycle in Detail** The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again. 48.1. **Defect Workflow** The diagram below shows the actual life cycle of a defect. 48.2. **Defect States**

- #1) **New:** This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) **Assigned:** In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) **Open:** Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.
- u) **Rejected:** If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- v) **Duplicate:** If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.
- w) **Deferred:** If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.
- x) **Not a Bug:** If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".
- #4) **Fixed:** When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".
- #5) **Pending Retest:** After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".
- #6) **Retest:** At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.
- #7) **Reopen:** If any issue persists in the defect, then it will be assigned to

the developer again for testing and the status of the defect gets changed to 'Reopen'. #8)

Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

48.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

49. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

50. Defect

Management 50.1. Defect Tracking In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier.

Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

50.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction. o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects. o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

50.1.2. Why is Defect Tracking

Necessary? o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process. o Monitors the bugs that had already been resolved, thereby saving time. o To ensure that right defects are being worked on. 50.2. Testing stages and

Saving time. 6 To ensure that right defects are being worked on. 30.2. Testing Stages and Defects Responsibilities SL No. Testing Stage Environment Defect Owner Defect Retested By Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team

Team S1 Team S1 Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team SIT Team SIT

4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager 50.5. Detect Image Calls Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more

occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that

focus the team on the “right” defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects

identified with every test.

50.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

50.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

50.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

50.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

50.5.1. Defect Trends by Release

- o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

50.5.2. Defects Density

- o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.
- o Defect Density = Total Defect/Size of the Release

50.5.3. Defect Fix Rate per Release

- o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .
- o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

50.5.4. Defect Aging

- o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

50.5.5. Defect Analysis through RCA

- o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.
- o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- o Fishbone diagram resembles the skeleton of a

fish with the problem forming the head of fish and causes forming the spine and bones of the fish. Follow the steps below to create a fishbone diagram: (21)Write the problem at the head of the fish. (22)Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N] (23)Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N. (24)Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

#### 50.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

##### 50.6.1. Action items, Action Owners, Tracking, Checklist etc.

51. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was resolved and recommendations for testing.
- References

52. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first?

Severity Description Examples

- 1 A critical incident with very high impact
- A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss
- 2 A major incident with significant impact
- A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted
- 3 A minor incident with low impact
- A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

Blocker —

Highest priority. Indicates that this issue takes precedence over all others. Critical — Indicates that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority.

53. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 54. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

55. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

56. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

57. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

57.1. Defect Workflow The diagram below shows the actual life cycle of a defect.

57.2. Defect States #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle. #2) Assigned: In this stage, a newly created

defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer. #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason. y) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer. z) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer. aa) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'. bb) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug". #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed". #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest". #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 57.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 58. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 59. Defect Management

#### 59.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

#### 59.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these

tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects. o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

### 59.1.2. Why is Defect Tracking Necessary?

o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.

o Monitors the bugs that had already been resolved, thereby saving time.

o To ensure that right defects are being worked on.

### 59.2. Testing stages and defects Responsibilities

SL No. Testing Stage Environment Defect Owner Defect Retested By Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team 4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager

### 59.3. Defect Triage Calls

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

#### 59.3.1. Thumb rules of Triaging

o All reported defects have been reviewed

o All accepted defects have been prioritized

o All accepted defects have severity attached

o All rejected defects should have reasonable explanations for the testing team

o Every defect has been assigned to an appropriate owner, individual or team

o Root cause analysis for every accepted defect has been done

#### 59.3.2. How does Defect Triaging work?

The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

### 59.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

### 59.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

#### 59.5.1. Defect Trends by Release

o The Defect Trends

report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

59.5.2. Defect Density o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

59.5.3. Defect Fix Rate per Release o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

59.5.4. Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

59.5.5. Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
- Follow the steps below to create a fishbone diagram:
  - (25) Write the problem at the head of the fish.
  - (26) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
  - (27) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
  - (28) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement.

Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit.

The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes.

Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

59.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects.

Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

59.6.1. Action items, Action Owners, Tracking, Checklist etc.

60. Defect Data

- Name of the Person
- Types of

Testing • Problem Summary • Detailed Description of Defect. • Steps to Reproduce • Life Cycle Phase • Work product where Defect was introduced. • Severity and Priority • Subsystem or Component where the Defect is introduced. • Project Activity occurs when the Defect is introduced. • Identification Method • Type of Defect • Projects and Products in which problems exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References

61. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples

1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss

2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted

3 A minor incident with low impact • A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

Blocker — Highest priority. Indicates that this issue takes precedence over all others.

Critical — Indicates that this issue is causing a problem and requires urgent attention.

Major — Indicates that this issue has a significant impact.

Minor — Indicates that this issue has a relatively minor impact.

Trivial — Lowest priority.

62. Defect Fix timeline

Severity Response Time Resolution Time 1 2 3 4

63. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process

Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

64. What is A Defect?

A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

65. Defect Management Life Cycle

When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if

the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

### 66. Defect Life Cycle in Detail

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

#### 66.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

#### 66.2. Defect States

#1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.

#2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.

#3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

cc) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.

dd) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.

ee) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.

ff) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".

#4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".

#5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".

#6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.

#7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.

#8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.

#9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

#### 66.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a

defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily. • The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

67. Goals of Defect Management Process (DMP) Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

68. Defect Management

68.1. Defect Tracking In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

68.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

68.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

68.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Defect Retested By
1	Unit Testing (UT)	DEV	DEV Team	Dev Team
2	System Testing (ST)	ST	ST Team	Team ST
3	System Integration Testing (SIT)	SIT	SIT Team	SIT Team
4	User Acceptance Testing (UAT)	UAT	UAT Team	UAT Team

68.3. Defect Triage

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the “right” defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

68.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

68.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

68.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

68.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

68.5.1. Defect Trends by Release

- o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

68.5.2. Defects Density

- o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.
- o Defect Density = Total Defect/Size of the Release

68.5.3. Defect Fix Rate per Release

- o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .
- o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

68.5.4. Defect Aging

- o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

68.5.5. Defect Analysis through RCA

- o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.
- o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.
- Steps to create a Fishbone Diagram:
  - Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
  - Follow the steps below to create a fishbone diagram:
    - (29) Write the problem at the head of the fish.
    - (30) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
    - (31) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
    - (32) Extend the causes to secondary, tertiary, and more levels as applicable.
- o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement.

Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of

problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

### 68.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

#### 68.6.1. Action items, Action Owners, Tracking, Checklist etc.

69. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was resolved and recommendations for testing.
- References

#### 70. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first?

Severity Description Examples

- 1 A critical incident with very high impact
- 2 A customer-facing service, like Jira Cloud, is down for all customers
- 3 Confidentiality or privacy is breached
- 4 Customer data loss
- 5 A major incident with significant impact
- 6 A customer-facing service is unavailable for a subset of customers
- 7 Core functionality is significantly impacted
- 8 A minor incident with low impact
- 9 A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

#### 71. Defect Fix timeline

Severity Response Time Resolution Time

Severity	Response Time	Resolution Time
1	1	2
2	2	3
3	4	7
4	7	2
5	2	1

#### 72. Conclusion

This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost

effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

73. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

74. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

75. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

75.1. Defect Workflow The diagram below shows the actual life cycle of a defect.

75.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

gg)

Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.

hh) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.

ii) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which

ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

1. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

2. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

3. Defect Life Cycle in Detail

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

3.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

3.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

a) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.

b) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.

c) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.

d) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the

defect gets changed to "Not a Bug". #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed". #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest". #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 3.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 4. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 5. Defect Management

#### 5.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

##### 5.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

##### 5.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

#### 5.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Defect Retested By
1	Unit Testing (UT)	DEV Team	Dev Team	System Testing (ST)
2	System Testing (ST)	ST Team	ST Team	System Integration Testing (SIT)
3	SIT	SIT Team	SIT Team	SIT Team
4	User Acceptance Testing (UAT)	UAT Team	UAT Team	Test Manager

### 5.3. Defect Triage Calls

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and

tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the “right” defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

5.3.1.Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

5.3.2.How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

5.4.Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

5.5.Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

5.5.1.Defect Trends by Release

The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

5.5.2.Defects Density

o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

$$\text{o Defect Density} = \frac{\text{Total Defect}}{\text{Size of the Release}}$$

5.5.3.Defect Fix Rate per Release

o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

5.5.4.Defect Aging

o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

5.5.5.Defect Analysis through RCA

o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software

Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization. o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.

Follow the steps below to create a fishbone diagram: (1) Write the problem at the head of the fish. (2) Identify the category of causes and write at end of each bone [cause category 1,

cause category 2 ..... cause category N] (3) Identify the primary causes under each category

and mark it as primary cause 1, primary cause 2, primary cause N. (4) Extend the causes to

secondary, tertiary, and more levels as applicable. o Parato charts to do the affinity analysis A

Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative

impact. Pareto Charts are useful to find the defects to prioritize in order to observe the

greatest overall improvement Pareto Analysis is a simple decision-making technique for

assessing competing problems and measuring the impact of fixing them. This allows you to

focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the

Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent

of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of

problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem

areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.

- Helping people to organize their workloads more effectively.

- Improving productivity.

- Improving profitability.

Pareto Analysis Steps 1. Identify and List Problems 2. Identify the Root Cause of Each Problem 3. Score Problems 4.

Group Problems Together 5. Add up Scores for Each Group 6. Take Action 5.6.Defect

Prevention Processes & Improvements Defect prevention is a framework and ongoing process

of collecting the defect data, doing root cause analysis, determining and implementing the

corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is

a crucial step or activity in any software development process and as can be seen from the

diagram below is pretty much half of our testing tasks: #1) Review and Inspection: This

method includes the review by an individual team member (self-checking), peer reviews and

inspection of all work products. #2) Walkthrough: This is more or less like a review but it's

mostly related to comparing the system to the prototype which will give a better idea

regarding the correctness and/or the look and-feel of the system. #3) Defect Logging and

Documentation: This method provides some key information, arguments/parameters that can

be used to support analyzing defects. #4) Root Cause Analysis: Root cause analysis includes

two major approaches: I) Pareto Analysis and II) Fishbone Analysis 5.6.1.Action items, Action

Owners, Tracking, Checklist etc. 6. Defect Data • Name of the Person • Types of Testing •

Problem Summary • Detailed Description of Defect. • Steps to Reproduce • Life Cycle Phase •

Work product where Defect was introduced. • Severity and Priority • Subsystem or Component

where the Defect is introduced. • Project Activity occurs when the Defect is introduced. •

Identification Method • Type of Defect • Projects and Products in which problems exist •

Current Owner • Current State of the Report • Work product where Defect occurred. • Impact

on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.

- Dates when various defect lifecycle phases occur. • Description of how the defect was

resolved and recommendations for testing. • References 7. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how

quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does

an incident have on users? Does it take down their whole system? Keep them from completing

a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand,

is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted 3 A minor incident with low impact • A minor inconvenience to customers, workaround available In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs Blocker — Highest priority. Indicates that this issue takes precedence over all others. Critical — Indicates that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority. 8. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 9. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process. 10. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 11. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project. 12. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects

from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

### 12.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

### 12.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.
- e) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- f) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.
- g) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.
- h) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".
- #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".
- #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".
- #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.
- #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.
- #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.
- #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 12.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 13. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 14. Defect Management

#### 14.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of

time and to effectively manage them, defect tracking system is used to make the job easier.

Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

14.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

14.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

14.2. Testing stages and defects Responsibilities SL No. Testing Stage Environment Defect Owner Defect Retested By Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team 4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager

14.3. Defect Triage Calls Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

14.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

14.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

14.4. Defect Metrics Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

14.5. Defect Reporting Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for

feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

14.5.1. Defect Trends by Release o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

14.5.2. Defects Density o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

14.5.3. Defect Fix Rate per Release o Defect removal efficiency =  $(1 - (\text{total defects caught by customer/ total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

14.5.4. Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

14.5.5. Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish. Follow the steps below to create a fishbone diagram:

- (5) Write the problem at the head of the fish.
- (6) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
- (7) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
- (8) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

14.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's

mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.

#3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.

#4) Root Cause Analysis: Root cause analysis includes two major approaches:

- I) Pareto Analysis and II) Fishbone Analysis

14.6.1. Action items, Action Owners, Tracking, Checklist etc.

15. Defect Data • Name of the Person • Types of Testing • Problem Summary • Detailed Description of Defect. • Steps to Reproduce • Life Cycle Phase • Work product where Defect was introduced. • Severity and Priority • Subsystem or Component where the Defect is introduced. • Project Activity occurs when the Defect is introduced. • Identification Method • Type of Defect • Projects and Products in which problems exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References

16. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first?

Severity Description Examples

- 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss
- 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted
- 3 A minor incident with low impact • A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

17. Defect Fix timeline

Severity	Response Time	Resolution Time
1	2	3
4	18	

18. Conclusion

This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free.

The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form.

Defect Management Process

Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

19. What is A Defect?

A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

20. Defect Management Life Cycle

When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing

team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

### 21. Defect Life Cycle in Detail

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

#### 21.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

#### 21.2. Defect States

#1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.

#2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.

#3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

- i) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- j) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.
- k) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.
- l) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".

#4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".

#5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".

#6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.

#7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.

#8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.

#9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

#### 21.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines

can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

## 22. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

## 23. Defect Management

### 23.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

#### 23.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

#### 23.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

### 23.2. Testing stages and defects Responsibilities

SL No. Testing Stage Environment Defect Owner Defect Retested By Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team 4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager

### 23.3. Defect Triage

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the “right” defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

#### 23.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

#### 23.3.2. How does Defect Triaging work?

The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and

Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

### 23.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

### 23.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

#### 23.5.1. Defect Trends by Release

- o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

#### 23.5.2. Defects Density

- o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.
- o Defect Density = Total Defect/Size of the Release

#### 23.5.3. Defect Fix Rate per Release

- o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .
- o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

#### 23.5.4. Defect Aging

- o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

#### 23.5.5. Defect Analysis through RCA

- o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.
- o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish. Follow the steps below to create a fishbone diagram:
- (9) Write the problem at the head of the fish.
- (10) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
- (11) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
- (12) Extend the causes to secondary, tertiary, and more levels as applicable.

#### o Pareto charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative

impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement. Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

### 23.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

#### 23.6.1. Action items, Action Owners, Tracking, Checklist etc.

24. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was resolved and recommendations for testing.
- References

### 25. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first?

Severity Description Examples

- 1 A critical incident with very high impact
- A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss

- 2 A major incident with significant impact
- A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted

- 3 A minor incident with low impact
- A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

### 26. Defect Fix timeline

Severity Response Time Resolution Time

Severity	Response Time	Resolution Time
Blocker	1	2
Critical	2	3
Major	3	4
Minor	4	5
Trivial	5	6

### 27. Conclusion

This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software

development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

28. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

29. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

30. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

30.1. Defect Workflow The diagram below shows the actual life cycle of a defect.

30.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.
- m) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- n) Duplicate: If the developer finds the defect as same as any

other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer. o) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'. p) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug". #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed". #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest". #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 30.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 31. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 32. Defect Management

#### 32.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

##### 32.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

##### 32.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

#### 32.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Retested By
1	Unit Testing	Unit Test Environment	Developer	QA Engineer
2	Integration Testing	Integration Test Environment	QA Engineer	QA Engineer
3	System Testing	System Test Environment	QA Engineer	QA Engineer
4	Acceptance Testing	Acceptance Test Environment	QA Engineer	QA Engineer
5	UAT Testing	UAT Test Environment	QA Engineer	QA Engineer

Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team 4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager 32.3. Defect Triage Calls Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the “right” defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test. 32.3.1. Thumb rules of Triaging o All reported defects have been reviewed o All accepted defects have been prioritized o All accepted defects have severity attached o All rejected defects should have reasonable explanations for the testing team o Every defect has been assigned to an appropriate owner, individual or team o Root cause analysis for every accepted defect has been done 32.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are: • The validity of the defect • Time sensitivity for resolution • The complexity involved in the resolution • Business impact This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced. 32.4. Defect Metrics Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be: • Defect distribution by cause • Defect distribution by feature/functional area • Defect distribution by Severity • Defect distribution by Priority • Defect distribution by type • Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user 32.5. Defect Reporting Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them. 32.5.1. Defect Trends by Release o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed. 32.5.2. Defects Density o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software. o Defect Density = Total Defect/Size of the Release 32.5.3. Defect Fix Rate per Release o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ . o Residual defect density = (total

number of defects found by a customer)/ (Total number of defects including customer found defects) x 100.

32.5.4. Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

32.5.5. Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish. Follow the steps below to create a fishbone diagram:

- (13) Write the problem at the head of the fish.
- (14) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
- (15) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
- (16) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement

Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit.

The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes.

Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

32.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects.

Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

32.6.1. Action items, Action Owners, Tracking, Checklist etc.

33. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was

resolved and recommendations for testing.

- References 34. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted 3 A minor incident with low impact • A minor inconvenience to customers, workaround available In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs Blocker — Highest priority. Indicates that this issue takes precedence over all others. Critical — Indicates that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority.

35. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 36. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

37. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

38. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different

states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

### 39. Defect Life Cycle in Detail

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

#### 39.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

#### 39.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.
- q) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- r) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.
- s) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.
- t) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".
- #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".
- #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".
- #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.
- #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.
- #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.
- #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

#### 39.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

#### 40. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

#### 41. Defect

**Management 41.1. Defect Tracking** In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

**41.1.1. Objectives of Defect Tracking** A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

**41.1.2. Why is Defect Tracking Necessary?** o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.

o Monitors the bugs that had already been resolved, thereby saving time.

o To ensure that right defects are being worked on.

**41.2. Testing stages and defects Responsibilities** SL No. Testing Stage Environment Defect Owner Defect Retested By

Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST

Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team

4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager

**41.3. Defect Triage Calls** Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

**41.3.1. Thumb rules of Triaging** o All reported defects have been reviewed

o All accepted defects have been prioritized

o All accepted defects have severity attached

o All rejected defects should have reasonable explanations for the testing team

o Every defect has been assigned to an appropriate owner, individual or team

o Root cause analysis for every accepted defect has been done

**41.3.2. How does Defect Triaging work?** The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution

- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

**41.4. Defect Metrics** Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs

have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

41.5. Defect Reporting Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

41.5.1. Defect Trends by Release o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

41.5.2. Defects Density o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

41.5.3. Defect Fix Rate per Release o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

41.5.4. Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

41.5.5. Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish. Follow the steps below to create a fishbone diagram:

- (17) Write the problem at the head of the fish.
- (18) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
- (19) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
- (20) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement. Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

41.6. Defect Prevention Processes & Improvements Defect prevention is a framework and ongoing process

of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

41.6.1. Action items, Action Owners, Tracking, Checklist etc.

42. Defect Data • Name of the Person • Types of Testing • Problem Summary • Detailed Description of Defect. • Steps to Reproduce • Life Cycle Phase • Work product where Defect was introduced. • Severity and Priority • Subsystem or Component where the Defect is introduced. • Project Activity occurs when the Defect is introduced. • Identification Method • Type of Defect • Projects and Products in which problems exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References

43. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first?

Severity Description Examples

- 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached • Customer data loss
- 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted
- 3 A minor incident with low impact • A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

44. Defect Fix timeline

Severity	Response Time	Resolution Time
1	1	2
2	2	3

45. Conclusion

This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

46. What is A Defect?

A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the

responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 47.

**Defect Management Life Cycle** When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project. 48. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again. 48.1. Defect Workflow The diagram below shows the actual life cycle of a defect. 48.2. Defect States #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle. #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer. #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason. u) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer. v) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer. w) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'. x) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug". #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed". #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest". #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8)

Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

48.3. Guidelines for Implementing a Defect Life Cycle Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

49. Goals of Defect Management Process (DMP) Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

50. Defect Management

50.1. Defect Tracking In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

50.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

50.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

50.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Defect Retested By
1	Unit Testing (UT)	DEV	DEV Team	Dev Team
2	System Testing (ST)	ST	ST Team	ST Team
3	System Integration Testing (SIT)	SIT	SIT Team	SIT Team
4	User Acceptance Testing (UAT)	UAT	UAT Team	UAT Team

50.3. Defect Triage

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

50.3.1. Thumb rules of Triaging

- o All reported defects have been

reviewed o All accepted defects have been prioritized o All accepted defects have severity attached o All rejected defects should have reasonable explanations for the testing team o Every defect has been assigned to an appropriate owner, individual or team o Root cause analysis for every accepted defect has been done 50.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

50.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

50.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

50.5.1. Defect Trends by Release

o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

50.5.2. Defects Density

o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

50.5.3. Defect Fix Rate per Release

o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

50.5.4. Defect Aging

o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

50.5.5. Defect Analysis through RCA

o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram

A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the

fish. Follow the steps below to create a fishbone diagram: (21)Write the problem at the head of the fish. (22)Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N] (23)Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N. (24)Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

#### 50.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

##### 50.6.1. Action items, Action Owners, Tracking, Checklist etc.

51. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was resolved and recommendations for testing.
- References

52. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples

- 1 A critical incident with very high impact
- A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss
- 2 A major incident with significant impact
- A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted
- 3 A minor incident with low impact
- A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

Blocker — Highest priority. Indicates that this issue takes precedence over all others.

Critical — Indicates

that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority.

53. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 54. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

55. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

56. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

57. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

57.1. Defect Workflow The diagram below shows the actual life cycle of a defect.

57.2. Defect States #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle. #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA

or test lead of the testing team to a developer. #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason. y) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer. z) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer. aa) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'. bb) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug". #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed". #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest". #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

57.3. Guidelines for Implementing a Defect Life Cycle Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

58. Goals of Defect Management Process (DMP) Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

59. Defect Management

59.1. Defect Tracking In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

59.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to

avoid further defects. o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

### 59.1.2. Why is Defect Tracking Necessary?

o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.

o Monitors the bugs that had already been resolved, thereby saving time.

o To ensure that right defects are being worked on.

### 59.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Defect Retested By
1	Unit Testing (UT)	DEV	DEV Team	Dev Team
2	System Testing (ST)	ST	ST Team	ST Team
3	System Integration Testing (SIT)	SIT	SIT Team	SIT Team
4	User Acceptance Testing (UAT)	UAT	UAT Team	UAT Team

### 59.3. Defect Triage Calls

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

#### 59.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

#### 59.3.2. How does Defect Triaging work?

The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

### 59.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

### 59.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

#### 59.5.1. Defect Trends by Release

- o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and

closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

59.5.2. Defect Density o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

59.5.3. Defect Fix Rate per Release o Defect removal efficiency = (1 - (total defects caught by customer/ total number of defects)) x 100.

o Residual defect density = (total number of defects found by a customer)/ (Total number of defects including customer found defects) x 100.

59.5.4. Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

59.5.5. Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.

Follow the steps below to create a fishbone diagram:

- (25) Write the problem at the head of the fish.
- (26) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
- (27) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
- (28) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement.

Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit.

The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes.

Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

59.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects.

Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

59.6.1. Action items, Action Owners, Tracking, Checklist etc.

60. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect
- Steps to Reproduce
- Life Cycle

Phase • Work product where Defect was introduced. • Severity and Priority • Subsystem or Component where the Defect is introduced. • Project Activity occurs when the Defect is introduced. • Identification Method • Type of Defect • Projects and Products in which problems exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References

61. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples

- 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss
- 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted
- 3 A minor incident with low impact • A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

62. Defect Fix timeline

Severity Response Time Resolution Time 1 2 3

4 63. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

64. What is A Defect?

A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

65. Defect Management Life Cycle

When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the

status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

#### 66. Defect Life Cycle in Detail

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

##### 66.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

##### 66.2. Defect States

#1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.

#2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.

#3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

cc) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.

dd) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.

ee) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.

ff) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".

#4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".

#5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".

#6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.

#7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.

#8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.

#9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

##### 66.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the

status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily. • The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

67. Goals of Defect Management Process (DMP) Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

68. Defect Management

68.1. Defect Tracking In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

68.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

68.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

68.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Defect Retested By
1	Unit Testing (UT)	DEV	DEV Team	Dev Team
2	System Testing (ST)	ST Team	ST Team	ST Team
3	System Integration Testing (SIT)	SIT Team	SIT Team	SIT Team
4	User Acceptance Testing (UAT)	UAT Team	UAT Team	Test Manager

68.3. Defect Triage Calls

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

68.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

68.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all

relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

68.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

68.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

68.5.1. Defect Trends by Release

- o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

68.5.2. Defects Density

- o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.
- o Defect Density = Total Defect/Size of the Release

68.5.3. Defect Fix Rate per Release

- o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .
- o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

68.5.4. Defect Aging

- o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

68.5.5. Defect Analysis through RCA

- o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.
- o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
- Follow the steps below to create a fishbone diagram:

- (29) Write the problem at the head of the fish.
- (30) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
- (31) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
- (32) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement. Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem

areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

68.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

68.6.1. Action items, Action Owners, Tracking, Checklist etc.

69. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was resolved and recommendations for testing.
- References

70. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first?

Severity Description Examples

- 1 A critical incident with very high impact
- A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss

- 2 A major incident with significant impact
- A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted

- 3 A minor incident with low impact
- A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

71. Defect Fix timeline

Severity Response Time Resolution Time

Severity	Response Time	Resolution Time
Blocker	1	2
Critical	2	3
Major	3	4
Minor	4	72.

Conclusion

This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The

organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process. 73. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 74.

Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project. 75. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again. 75.1. Defect Workflow The diagram below shows the actual life cycle of a defect. 75.2.

Defect States #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle. #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer. #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason. gg)

Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer. hh) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer. ii) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and

effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

1. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

2. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

3. Defect Life Cycle in Detail

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

3.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

3.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

a) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.

b) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.

c) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.

d) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".

#4) Fixed: When the developer finishes the task of fixing

a defect by making the required changes then they can mark the status of the defect as "Fixed". #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest". #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 3.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 4. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 5. Defect Management

#### 5.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

##### 5.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

##### 5.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

#### 5.2. Testing stages and defects

Responsibilities SL No. Testing Stage Environment Defect Owner Defect Retested By Defect Tracking By

1	Unit Testing (UT)	DEV DEV Team	Dev Team	2	System Testing (ST)	ST ST Team	ST Team	ST Team	3	System Integration Testing (SIT)	SIT SIT Team	SIT Team	SIT Team	4	User Acceptance Testing (UAT)	UAT UAT Team	UAT Team	Test Manager
---	-------------------	--------------	----------	---	---------------------	------------	---------	---------	---	----------------------------------	--------------	----------	----------	---	-------------------------------	--------------	----------	--------------

### 5.3. Defect Triage Calls

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized

projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

5.3.1.Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

5.3.2.How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

5.4.Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

5.5.Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

5.5.1.Defect Trends by Release

- o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

5.5.2.Defects Density

- o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.
- o Defect Density = Total Defect/Size of the Release

5.5.3.Defect Fix Rate per Release

- o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .
- o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

5.5.4.Defect Aging

- o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

5.5.5.Defect Analysis through RCA

- o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the

deliverables and the processes, not only at the team level but also across the organization. o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.

Follow the steps below to create a fishbone diagram: (1) Write the problem at the head of the fish. (2) Identify the category of causes and write at end of each bone [cause category 1,

cause category 2 ..... cause category N] (3) Identify the primary causes under each category

and mark it as primary cause 1, primary cause 2, primary cause N. (4) Extend the causes to

secondary, tertiary, and more levels as applicable. o Parato charts to do the affinity analysis A

Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative

impact. Pareto Charts are useful to find the defects to prioritize in order to observe the

greatest overall improvement Pareto Analysis is a simple decision-making technique for

assessing competing problems and measuring the impact of fixing them. This allows you to

focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the

Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent

of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of

problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem

areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.

- Helping people to organize their workloads more effectively.

- Improving productivity.

- Improving profitability.

Pareto Analysis Steps 1. Identify and List Problems 2. Identify the Root Cause of Each Problem 3. Score Problems 4.

Group Problems Together 5. Add up Scores for Each Group 6. Take Action 5.6.Defect

Prevention Processes & Improvements Defect prevention is a framework and ongoing process

of collecting the defect data, doing root cause analysis, determining and implementing the

corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is

a crucial step or activity in any software development process and as can be seen from the

diagram below is pretty much half of our testing tasks: #1) Review and Inspection: This

method includes the review by an individual team member (self-checking), peer reviews and

inspection of all work products. #2) Walkthrough: This is more or less like a review but it's

mostly related to comparing the system to the prototype which will give a better idea

regarding the correctness and/or the look and-feel of the system. #3) Defect Logging and

Documentation: This method provides some key information, arguments/parameters that can

be used to support analyzing defects. #4) Root Cause Analysis: Root cause analysis includes

two major approaches: I) Pareto Analysis and II) Fishbone Analysis 5.6.1.Action items, Action

Owners, Tracking, Checklist etc. 6. Defect Data • Name of the Person • Types of Testing •

Problem Summary • Detailed Description of Defect. • Steps to Reproduce • Life Cycle Phase •

Work product where Defect was introduced. • Severity and Priority • Subsystem or Component

where the Defect is introduced. • Project Activity occurs when the Defect is introduced. •

Identification Method • Type of Defect • Projects and Products in which problems exist •

Current Owner • Current State of the Report • Work product where Defect occurred. • Impact

on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.

- Dates when various defect lifecycle phases occur. • Description of how the defect was

resolved and recommendations for testing. • References 7. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how

quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does

an incident have on users? Does it take down their whole system? Keep them from completing

a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand,

is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to

be fixed first? Severity Description Examples 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted 3 A minor incident with low impact • A minor inconvenience to customers, workaround available In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs Blocker — Highest priority. Indicates that this issue takes precedence over all others. Critical — Indicates that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority. 8. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 9. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process. 10. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 11. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project. 12. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as

any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

### 12.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

### 12.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.
- e) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- f) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.
- g) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.
- h) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".
- #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".
- #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".
- #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.
- #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.
- #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.
- #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 12.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 13. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 14. Defect Management

#### 14.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier.

Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

14.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

14.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

14.2. Testing stages and defects Responsibilities SL No. Testing Stage Environment Defect Owner Defect Retested By

Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team

4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager

14.3. Defect Triage Calls Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

14.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team

o Every defect has been assigned to an appropriate owner, individual or team

o Root cause analysis for every accepted defect has been done

14.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

14.4. Defect Metrics Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters.

Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority

- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

14.5. Defect Reporting Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team

checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

14.5.1. Defect Trends by Release o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

14.5.2. Defects Density o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

14.5.3. Defect Fix Rate per Release o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

14.5.4. Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

14.5.5. Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish. Follow the steps below to create a fishbone diagram:

- (5) Write the problem at the head of the fish.
- (6) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
- (7) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
- (8) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

14.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea

regarding the correctness and/or the look and-feel of the system. #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects. #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

#### 14.6.1. Action items, Action Owners, Tracking, Checklist etc.

15. Defect Data • Name of the Person • Types of Testing • Problem Summary • Detailed Description of Defect. • Steps to Reproduce • Life Cycle Phase • Work product where Defect was introduced. • Severity and Priority • Subsystem or Component where the Defect is introduced. • Project Activity occurs when the Defect is introduced. • Identification Method • Type of Defect • Projects and Products in which problems exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References

#### 16. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples

- 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached
- 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted
- 3 A minor incident with low impact • A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

#### 17. Defect Fix timeline

Severity Response Time Resolution Time

1	2	3
4	18. Conclusion	This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.
19. What is A Defect?	A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.	
20. Defect Management Life Cycle	When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is	

different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

### 21. Defect Life Cycle in Detail

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

#### 21.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

#### 21.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.
- i) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- j) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.
- k) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.
- l) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".
- #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".
- #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".
- #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.
- #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.
- #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.
- #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

#### 21.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is

very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).

- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

## 22. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

## 23. Defect Management

### 23.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

#### 23.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

#### 23.1.2. Why is Defect Tracking Necessary?

- A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- Monitors the bugs that had already been resolved, thereby saving time.
- To ensure that right defects are being worked on.

## 23.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Defect Retested By
1	Unit Testing (UT)	DEV	DEV Team	Dev Team
2	System Testing (ST)	ST	ST Team	ST Team
3	Integration Testing (SIT)	SIT	SIT Team	SIT Team
4	User Acceptance Testing (UAT)	UAT	UAT Team	UAT Team
			Test Manager	

## 23.3. Defect Triage

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

#### 23.3.1. Thumb rules of Triaging

- All reported defects have been reviewed
- All accepted defects have been prioritized
- All accepted defects have severity attached
- All rejected defects should have reasonable explanations for the testing team
- Every defect has been assigned to an appropriate owner, individual or team
- Root cause analysis for every accepted defect has been done

#### 23.3.2. How does Defect Triaging work?

The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach

priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

### 23.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

### 23.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

#### 23.5.1. Defect Trends by Release

- o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

#### 23.5.2. Defects Density

- o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.
- o Defect Density = Total Defect/Size of the Release

#### 23.5.3. Defect Fix Rate per Release

- o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .
- o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

#### 23.5.4. Defect Aging

- o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

#### 23.5.5. Defect Analysis through RCA

- o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.
- o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.
- Steps to create a Fishbone Diagram:
  - Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
  - Follow the steps below to create a fishbone diagram:
  - (9) Write the problem at the head of the fish.
  - (10) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
  - (11) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
  - (12) Extend the causes to secondary, tertiary, and more levels as applicable.
- o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the

greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

### 23.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

#### 23.6.1. Action items, Action Owners, Tracking, Checklist etc.

#### 24. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was resolved and recommendations for testing.
- References

#### 25. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first?

Severity Description Examples

- 1 A critical incident with very high impact
- A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss

- 2 A major incident with significant impact
- A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted

- 3 A minor incident with low impact
- A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

#### 26. Defect Fix timeline

Severity Response Time Resolution Time

1	2	3
---	---	---

#### 27. Conclusion

This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and

development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

28. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

29. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

30. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

30.1. Defect Workflow The diagram below shows the actual life cycle of a defect.

30.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.
- m) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- n) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the

defect is changed to 'Duplicate' by the developer. o) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'. p) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug". #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed". #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest". #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 30.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 31. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 32. Defect Management

#### 32.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

##### 32.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

##### 32.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

#### 32.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Defect Retested By
1	Unit Testing (UT)	DEV	DEV Team	Dev Team
2	System Testing (ST)	ST	ST	ST

Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team 4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager 32.3. Defect Triage Calls Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test. 32.3.1. Thumb rules of Triaging o All reported defects have been reviewed o All accepted defects have been prioritized o All accepted defects have severity attached o All rejected defects should have reasonable explanations for the testing team o Every defect has been assigned to an appropriate owner, individual or team o Root cause analysis for every accepted defect has been done 32.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are: • The validity of the defect • Time sensitivity for resolution • The complexity involved in the resolution • Business impact This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced. 32.4. Defect Metrics Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be: • Defect distribution by cause • Defect distribution by feature/functional area • Defect distribution by Severity • Defect distribution by Priority • Defect distribution by type • Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user 32.5. Defect Reporting Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them. 32.5.1. Defect Trends by Release o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed. 32.5.2. Defects Density o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software. o Defect Density = Total Defect/Size of the Release 32.5.3. Defect Fix Rate per Release o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ . o Residual defect density = (total number of defects found by a customer) / (Total number of defects including customer found)

defects) x 100.

32.5.4. Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

32.5.5. Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
- Follow the steps below to create a fishbone diagram:

- (13) Write the problem at the head of the fish.
- (14) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
- (15) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
- (16) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement

Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit.

The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes.

Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

32.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects.

Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

32.6.1. Action items, Action Owners, Tracking, Checklist etc.

33. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was resolved and recommendations for testing.
- References

34. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted 3 A minor incident with low impact • A minor inconvenience to customers, workaround available In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs Blocker — Highest priority. Indicates that this issue takes precedence over all others. Critical — Indicates that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority.

35. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 36. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

37. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

38. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its

lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

### 39. Defect Life Cycle in Detail

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

#### 39.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

#### 39.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.
- q) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- r) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.
- s) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.
- t) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".
- #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".
- #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".
- #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.
- #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.
- #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.
- #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 39.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 40. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 41. Defect Management

#### 41.1. Defect Tracking

In software engineering, defect tracking is the process of

tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

41.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

41.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

41.2. Testing stages and defects Responsibilities SL No. Testing Stage Environment Defect Owner Defect Retested By Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team 4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager 41.3. Defect Triage Calls

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

41.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

41.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

41.4. Defect Metrics Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters.

Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

41.5. Defect Reporting Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

41.5.1. Defect Trends by Release o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

41.5.2. Defects Density o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

41.5.3. Defect Fix Rate per Release o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

41.5.4. Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

41.5.5. Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish. Follow the steps below to create a fishbone diagram:

- (17) Write the problem at the head of the fish.
- (18) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
- (19) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
- (20) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

41.6. Defect Prevention Processes & Improvements Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the

corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

41.6.1. Action items, Action Owners, Tracking, Checklist etc.

42. Defect Data • Name of the Person • Types of Testing • Problem Summary • Detailed Description of Defect. • Steps to Reproduce • Life Cycle Phase • Work product where Defect was introduced. • Severity and Priority • Subsystem or Component where the Defect is introduced. • Project Activity occurs when the Defect is introduced. • Identification Method • Type of Defect • Projects and Products in which problems exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References

43. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples

- 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss
- 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted
- 3 A minor incident with low impact • A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

44. Defect Fix timeline

Severity Response Time Resolution Time

1	2	3
---	---	---

45. Conclusion

This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

46. What is A Defect?

A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as

possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 47.

**Defect Management Life Cycle** When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project. 48. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again. 48.1. Defect Workflow The diagram below shows the actual life cycle of a defect. 48.2.

**Defect States #1) New:** This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle. #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer. #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason. u) **Rejected:** If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer. v) **Duplicate:** If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer. w) **Deferred:** If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'. x) **Not a Bug:** If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug". #4) **Fixed:** When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed". #5) **Pending Retest:** After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest". #6) **Retest:** At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. #7) **Reopen:** If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8) **Verified:** If the tester does not find any issue in the defect after being assigned to the

developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

48.3. Guidelines for Implementing a Defect Life Cycle Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

49. Goals of Defect Management Process (DMP) Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

50. Defect Management

50.1. Defect Tracking In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

50.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

50.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

50.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Retested By
1	Unit Testing (UT)	DEV	DEV Team	Dev Team
2	System Testing (ST)	ST	ST Team	ST Team
3	System Integration Testing (SIT)	SIT	SIT Team	SIT Team
4	User Acceptance Testing (UAT)	UAT	UAT Team	UAT Team

50.3. Defect Triage Calls

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

50.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity

attached o All rejected defects should have reasonable explanations for the testing team o Every defect has been assigned to an appropriate owner, individual or team o Root cause analysis for every accepted defect has been done 50.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

50.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

50.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

50.5.1. Defect Trends by Release

- o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

50.5.2. Defects Density

- o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.
- o Defect Density = Total Defect/Size of the Release

50.5.3. Defect Fix Rate per Release

- o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .
- o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

50.5.4. Defect Aging

- o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

50.5.5. Defect Analysis through RCA

- o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.
- o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
- Follow the steps below to create a fishbone diagram:
- (21) Write the problem at the head

of the fish. (22) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N] (23) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N. (24) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

#### 50.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

#### 50.6.1. Action items, Action Owners, Tracking, Checklist etc.

51. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was resolved and recommendations for testing.
- References

52. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples

- 1 A critical incident with very high impact
- A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss

2 A major incident with significant impact

- A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted

3 A minor incident with low impact

- A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this

issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority. 53. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 54. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process. 55. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 56. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project. 57. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again. 57.1. Defect Workflow The diagram below shows the actual life cycle of a defect. 57.2. Defect States #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle. #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer. #3) Open: Here, the developer starts the

process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

- y) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- z) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.

- aa) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.
- bb) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".

- #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".

- #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".
- #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.
- #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.
- #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.
- #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 57.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 58. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 59. Defect Management

#### 59.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

#### 59.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of

the work. Also, it enhances the efficient work delivered.

### 59.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

### 59.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Defect Retested By
1	Unit Testing (UT)	DEV	DEV Team	Dev Team
2	System Testing (ST)	ST	ST Team	ST Team
3	System Integration Testing (SIT)	SIT	SIT Team	SIT Team
4	User Acceptance Testing (UAT)	UAT	UAT Team	UAT Team
			Test Manager	

### 59.3. Defect Triage

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the “right” defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

#### 59.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

#### 59.3.2. How does Defect Triaging work?

The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

### 59.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

### 59.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

#### 59.5.1. Defect Trends by Release

- o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend

at which the defects are processed.

59.5.2. Defect Density o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

59.5.3. Defect Fix Rate per Release o Defect removal efficiency = (1 - (total defects caught by customer/ total number of defects)) x 100.

o Residual defect density = (total number of defects found by a customer)/ (Total number of defects including customer found defects) x 100.

59.5.4. Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

59.5.5. Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.

Follow the steps below to create a fishbone diagram:

- (25) Write the problem at the head of the fish.
- (26) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
- (27) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
- (28) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement. Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

59.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

59.6.1. Action items, Action Owners, Tracking, Checklist etc.

60. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or

Component where the Defect is introduced. • Project Activity occurs when the Defect is introduced. • Identification Method • Type of Defect • Projects and Products in which problems exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References

61. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples

1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss

2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted

3 A minor incident with low impact • A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

Blocker — Highest priority. Indicates that this issue takes precedence over all others.

Critical — Indicates that this issue is causing a problem and requires urgent attention.

Major — Indicates that this issue has a significant impact.

Minor — Indicates that this issue has a relatively minor impact.

Trivial — Lowest priority.

62. Defect Fix timeline

Severity Response Time Resolution Time

1	2	3
---	---	---

4 63. Conclusion

This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process

Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

64. What is A Defect?

A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

65. Defect Management Life Cycle

When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all

defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

### 66. Defect Life Cycle in Detail

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

#### 66.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

#### 66.2. Defect States

#1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.

#2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.

#3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

cc) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.

dd) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.

ee) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.

ff) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".

#4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".

#5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".

#6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.

#7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.

#8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.

#9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

#### 66.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that

particular defect can understand the reason of such a status of a defect very easily. • The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 67. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 68. Defect Management

#### 68.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

#### 68.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

#### 68.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

#### 68.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Defect Retested By
1	Unit Testing (UT)	DEV	DEV Team	Dev Team
2	System Testing (ST)	ST	ST	Team
3	System Integration Testing (SIT)	SIT	SIT	Team
4	User Acceptance Testing (UAT)	UAT	UAT Team	UAT Team
				Test Manager

#### 68.3. Defect Triage

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the “right” defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

#### 68.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

#### 68.3.2. How does Defect Triaging work?

The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the

basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

#### 68.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

#### 68.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

##### 68.5.1. Defect Trends by Release

o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

##### 68.5.2. Defects Density

o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

##### 68.5.3. Defect Fix Rate per Release

o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

##### 68.5.4. Defect Aging

o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

##### 68.5.5. Defect Analysis through RCA

o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
- Follow the steps below to create a fishbone diagram:
  - (29) Write the problem at the head of the fish.
  - (30) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
  - (31) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
  - (32) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement.

Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

-

Identifying and prioritizing problems and tasks. • Helping people to organize their workloads more effectively. • Improving productivity. • Improving profitability. Pareto Analysis Steps 1. Identify and List Problems 2. Identify the Root Cause of Each Problem 3. Score Problems 4. Group Problems Together 5. Add up Scores for Each Group 6. Take Action 68.6. Defect Prevention Processes & Improvements Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks: #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products. #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system. #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects. #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis 68.6.1. Action items, Action Owners, Tracking, Checklist etc. 69. Defect Data • Name of the Person • Types of Testing • Problem Summary • Detailed Description of Defect. • Steps to Reproduce • Life Cycle Phase • Work product where Defect was introduced. • Severity and Priority • Subsystem or Component where the Defect is introduced. • Project Activity occurs when the Defect is introduced. • Identification Method • Type of Defect • Projects and Products in which problems exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References 70. Defect Priority and Severity Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted 3 A minor incident with low impact • A minor inconvenience to customers, workaround available In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs Blocker — Highest priority. Indicates that this issue takes precedence over all others. Critical — Indicates that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority. 71. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 72. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which

is collectively known as a Defect Management Process. 73. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 74. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project. 75. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again. 75.1. Defect Workflow The diagram below shows the actual life cycle of a defect. 75.2. Defect States #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle. #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer. #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason. gg) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer. hh) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer. ii) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects

found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

1. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

2. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

3. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

3.1. Defect Workflow The diagram below shows the actual life cycle of a defect.

3.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

a) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.

b) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.

c) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.

d) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".

#4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as

"Fixed". #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest". #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 3.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 4. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 5. Defect Management

#### 5.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

##### 5.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

##### 5.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

#### 5.2. Testing stages and defects

Responsibilities SL No. Testing Stage Environment Defect Owner Defect Retested By Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team 4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager

### 5.3. Defect Triage Calls

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the

team on the “right” defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

5.3.1.Thumb rules of Triaging o All reported defects have been reviewed o All accepted defects have been prioritized o All accepted defects have severity attached o All rejected defects should have reasonable explanations for the testing team o Every defect has been assigned to an appropriate owner, individual or team o Root cause analysis for every accepted defect has been done

5.3.2.How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

5.4.Defect Metrics Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

5.5.Defect Reporting Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

5.5.1.Defect Trends by Release o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

5.5.2.Defects Density o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

5.5.3.Defect Fix Rate per Release o Defect removal efficiency =  $(1 - (\text{total defects caught by customer/ total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

5.5.4.Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

5.5.5.Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.

Follow the steps below to create a fishbone diagram: (1) Write the problem at the head of the fish. (2) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N] (3) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N. (4) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement

Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
  - Helping people to organize their workloads more effectively.
  - Improving productivity.
  - Improving profitability.
- Pareto Analysis Steps 1. Identify and List Problems 2. Identify the Root Cause of Each Problem 3. Score Problems 4. Group Problems Together 5. Add up Scores for Each Group 6. Take Action

5.6.Defect Prevention Processes & Improvements Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks: #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products. #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system. #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects. #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

5.6.1.Action items, Action Owners, Tracking, Checklist etc. 6. Defect Data • Name of the Person • Types of Testing • Problem Summary • Detailed Description of Defect. • Steps to Reproduce • Life Cycle Phase • Work product where Defect was introduced. • Severity and Priority • Subsystem or Component where the Defect is introduced. • Project Activity occurs when the Defect is introduced. • Identification Method • Type of Defect • Projects and Products in which problems exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.

• Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References 7. Defect Priority and Severity Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples 1 A critical incident with very high impact • A

customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted 3 A minor incident with low impact • A minor inconvenience to customers, workaround available In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs Blocker — Highest priority. Indicates that this issue takes precedence over all others. Critical — Indicates that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority. 8. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 9. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process. 10. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 11. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project. 12. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect,

assuring that it won't get reproduced again.

### 12.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

### 12.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.

#2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.

#3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

e) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.

f) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.

g) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.

h) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".

#4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".

#5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".

#6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.

#7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.

#8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.

#9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

**12.3. Guidelines for Implementing a Defect Life Cycle**

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).

- Defect Life Cycle should be properly documented to avoid any confusion in the future.

- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.

- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.

- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

**13. Goals of Defect Management Process (DMP)**

Given below are the various goals of this process:

- Prevent the Defect

- Early Detection

- Minimize the impact

- Resolution of the Defect

- Process improvement

**14. Defect Management**

**14.1. Defect Tracking**

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier.

Defects are tracked based on various parameters such as:

- Defect Id

- Priority

- Severity

Created by • Created Date • Assigned to • Resolved Date • Resolved By • Status • Reason

14.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction. o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects. o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

14.1.2. Why is Defect Tracking Necessary? o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process. o Monitors the bugs that had already been resolved, thereby saving time. o To ensure that right defects are being worked on.

14.2. Testing stages and defects Responsibilities SL No. Testing Stage Environment Defect Owner Defect Retested By Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team 4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager

14.3. Defect Triage Calls Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

14.3.1. Thumb rules of Triaging o All reported defects have been reviewed o All accepted defects have been prioritized o All accepted defects have severity attached o All rejected defects should have reasonable explanations for the testing team o Every defect has been assigned to an appropriate owner, individual or team o Root cause analysis for every accepted defect has been done

14.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

14.4. Defect Metrics Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

14.5. Defect Reporting Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect

reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

14.5.1. Defect Trends by Release o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

14.5.2. Defects Density o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

14.5.3. Defect Fix Rate per Release o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

14.5.4. Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

14.5.5. Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
- Follow the steps below to create a fishbone diagram:

- (5) Write the problem at the head of the fish.
- (6) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
- (7) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
- (8) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement

Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit.

The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes.

Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

14.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects.

Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and

Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.

#4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

#### 14.6.1. Action items, Action Owners, Tracking, Checklist etc.

15. Defect Data • Name of the Person • Types of Testing • Problem Summary • Detailed Description of Defect. • Steps to Reproduce • Life Cycle Phase • Work product where Defect was introduced. • Severity and Priority • Subsystem or Component where the Defect is introduced. • Project Activity occurs when the Defect is introduced. • Identification Method • Type of Defect • Projects and Products in which problems exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References

#### 16. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples

- 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss
- 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted
- 3 A minor incident with low impact • A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

#### 17. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4

#### 18. Conclusion

This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

#### 19. What is A Defect?

A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

#### 20. Defect Management Life Cycle

When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software

Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

### 21. Defect Life Cycle in Detail

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

#### 21.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

#### 21.2. Defect States #1)

New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.

#2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.

#3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

- i) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- j) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.
- k) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.
- l) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".

#4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".

#5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".

#6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.

#7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.

#8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.

#9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

#### 21.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly

understands the different states of a defect (discussed above). • Defect Life Cycle should be properly documented to avoid any confusion in the future. • Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results. • Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily. • The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

## 22. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

## 23. Defect Management

### 23.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

#### 23.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

#### 23.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

### 23.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Defect Retested By
1	Unit Testing (UT)	DEV	DEV Team	Dev Team
2	System Testing (ST)	ST	ST Team	ST Team
3	System Integration Testing (SIT)	SIT	SIT Team	SIT Team
4	User Acceptance Testing (UAT)	UAT	UAT Team	UAT Team

### 23.3. Defect Triage

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the “right” defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

#### 23.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

#### 23.3.2. How does Defect Triaging work?

The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity

corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

### 23.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

### 23.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

#### 23.5.1. Defect Trends by Release

- o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

#### 23.5.2. Defects Density

- o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.
- o Defect Density = Total Defect/Size of the Release

#### 23.5.3. Defect Fix Rate per Release

- o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .
- o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

#### 23.5.4. Defect Aging

- o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

#### 23.5.5. Defect Analysis through RCA

- o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.
- o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.
- Steps to create a Fishbone Diagram:
  - Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
  - Follow the steps below to create a fishbone diagram:
    - (9) Write the problem at the head of the fish.
    - (10) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
    - (11) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
    - (12) Extend the causes to secondary, tertiary, and more levels as applicable.
  - o Pareto charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement
  - Pareto Analysis is a simple decision-making technique for

assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

### 23.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

#### 23.6.1. Action items, Action Owners, Tracking, Checklist etc.

24. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was resolved and recommendations for testing.
- References

25. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first?

Severity Description Examples

- 1 A critical incident with very high impact
- A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss

- 2 A major incident with significant impact
- A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted

- 3 A minor incident with low impact
- A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

26. Defect Fix timeline

Severity Response Time Resolution Time

- 1
- 2
- 3
- 4
27. Conclusion

This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best

form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process. 28. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 29.

Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get

increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project. 30. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again. 30.1. Defect Workflow The diagram below shows the actual life cycle of a defect. 30.2.

Defect States #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle. #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer. #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason. m) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer. n) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer. o) Deferred: If the developer feels that the

defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'. p) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug". #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed". #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest". #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

30.3. Guidelines for Implementing a Defect Life Cycle Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

31. Goals of Defect Management Process (DMP) Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

32. Defect Management

32.1. Defect Tracking In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

32.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

32.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

32.2. Testing stages and defects Responsibilities

SL No. Testing Stage Environment Defect Owner Defect Retested By

Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team

4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager 32.3. Defect Triage Calls Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the “right” defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

32.3.1. Thumb rules of Triaging o All reported defects have been reviewed o All accepted defects have been prioritized o All accepted defects have severity attached o All rejected defects should have reasonable explanations for the testing team o Every defect has been assigned to an appropriate owner, individual or team o Root cause analysis for every accepted defect has been done

32.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

32.4. Defect Metrics Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

32.5. Defect Reporting Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

32.5.1. Defect Trends by Release o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

32.5.2. Defects Density o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

- o Defect Density = Total Defect/Size of the Release

32.5.3. Defect Fix Rate per Release o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

- o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

32.5.4. Defect Aging o Defect Age (in Time) is the difference in time between

the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed). 32.5.5. Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization. o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms. Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish. Follow the steps below to create a fishbone diagram: (13)Write the problem at the head of the fish. (14)Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N] (15)Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N. (16)Extend the causes to secondary, tertiary, and more levels as applicable. o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

32.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

32.6.1. Action items, Action Owners, Tracking, Checklist etc.

33. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was resolved and recommendations for testing.
- References

34. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how

quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted 3 A minor incident with low impact • A minor inconvenience to customers, workaround available In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs Blocker — Highest priority. Indicates that this issue takes precedence over all others. Critical — Indicates that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority. 35. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 36. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process. 37. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 38. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from

the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

### 39. Defect Life Cycle in Detail

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

#### 39.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

#### 39.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.
- q) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- r) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.
- s) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.
- t) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".
- #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".
- #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".
- #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.
- #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.
- #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.
- #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

#### 39.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 40. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 41. Defect Management

#### 41.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by

inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier.

Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

41.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

41.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

41.2. Testing stages and defects Responsibilities SL No. Testing Stage Environment Defect Owner Defect Retested By Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team 4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager

41.3. Defect Triage Calls Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

41.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

41.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

41.4. Defect Metrics Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect

distribution by feature/functional area • Defect distribution by Severity • Defect distribution by Priority • Defect distribution by type • Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user 41.5. Defect Reporting Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

41.5.1. Defect Trends by Release o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

41.5.2. Defects Density o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

41.5.3. Defect Fix Rate per Release o Defect removal efficiency = (1 - (total defects caught by customer/ total number of defects)) x 100.

o Residual defect density = (total number of defects found by a customer)/ (Total number of defects including customer found defects) x 100.

41.5.4. Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

41.5.5. Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish. Follow the steps below to create a fishbone diagram:

- (17) Write the problem at the head of the fish.
- (18) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
- (19) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
- (20) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

41.6. Defect Prevention Processes & Improvements Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is

a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks: #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products. #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system. #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects. #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

41.6.1. Action items, Action Owners, Tracking, Checklist etc.

42. Defect Data • Name of the Person • Types of Testing • Problem Summary • Detailed Description of Defect. • Steps to Reproduce • Life Cycle Phase • Work product where Defect was introduced. • Severity and Priority • Subsystem or Component where the Defect is introduced. • Project Activity occurs when the Defect is introduced. • Identification Method • Type of Defect • Projects and Products in which problems exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References

43. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples

1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss

2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted

3 A minor incident with low impact • A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

Blocker — Highest priority. Indicates that this issue takes precedence over all others.

Critical — Indicates that this issue is causing a problem and requires urgent attention.

Major — Indicates that this issue has a significant impact.

Minor — Indicates that this issue has a relatively minor impact.

Trivial — Lowest priority.

44. Defect Fix timeline

Severity Response Time Resolution Time 1 2 3

45. Conclusion

This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

46. What is A Defect?

A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand

the defect life cycle before moving to the workflow and different states of the defect. 47. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project. 48. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again. 48.1. Defect Workflow The diagram below shows the actual life cycle of a defect. 48.2. Defect States #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle. #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer. #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason. u) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer. v) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer. w) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'. x) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug". #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed". #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest". #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status

of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

48.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

49. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

50. Defect Management

50.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

50.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

50.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

50.2. Testing stages and defects Responsibilities

SL No. Testing Stage Environment Defect Owner Defect Retested By Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team 4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager

50.3. Defect Triage

Calls Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

50.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team o

Every defect has been assigned to an appropriate owner, individual or team o Root cause analysis for every accepted defect has been done 50.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

#### 50.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

#### 50.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

##### 50.5.1. Defect Trends by Release

o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

##### 50.5.2. Defects Density

o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

##### 50.5.3. Defect Fix Rate per Release

o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

##### 50.5.4. Defect Aging

o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

##### 50.5.5. Defect Analysis through RCA

o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
- Follow the steps below to create a fishbone diagram:
  - (21) Write the problem at the head of the fish.
  - (22) Identify the category of causes and write at end of each bone [cause category]

1, cause category 2 ..... cause category N] (23) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N. (24) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement. Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

#### 50.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

#### 50.6.1. Action items, Action Owners, Tracking, Checklist etc.

51. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was resolved and recommendations for testing.
- References

52. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first?

Severity Description Examples

- 1 A critical incident with very high impact
- A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss

- 2 A major incident with significant impact
- A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted

- 3 A minor incident with low impact
- A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.

Trivial — Lowest priority. 53. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 54. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process. 55. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 56. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project. 57. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again. 57.1. Defect Workflow The diagram below shows the actual life cycle of a defect. 57.2. Defect States #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle. #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer. #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that

the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

- y) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- z) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.

- aa) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.
- bb) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".

- #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".

- #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".
- #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.
- #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.
- #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.
- #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 57.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 58. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 59. Defect Management

#### 59.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

#### 59.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

#### 59.1.2. Why is Defect Tracking

Necessary? o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process. o Monitors the bugs that had already been resolved, thereby saving time. o To ensure that right defects are being worked on.

59.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Defect Retested By
1	Unit Testing (UT)	DEV	DEV Team	Dev Team
2	System Testing (ST)	ST	ST	ST
3	System Integration Testing (SIT)	SIT	SIT	Team SIT
4	User Acceptance Testing (UAT)	UAT	UAT Team	UAT Team
				Test Manager

59.3. Defect Triage Calls

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the “right” defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

59.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

59.3.2. How does Defect Triaging work?

The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

59.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

59.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

59.5.1. Defect Trends by Release

- o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

59.5.2. Defects Density

- o Defect density can be defined as

the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

- o Defect Density = Total Defect/Size of the Release
- 59.5.3. Defect Fix Rate per Release
- o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .
- o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

59.5.4. Defect Aging

- o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

59.5.5. Defect Analysis through RCA

- o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.
- o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
- Follow the steps below to create a fishbone diagram:
- (25) Write the problem at the head of the fish.
- (26) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
- (27) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
- (28) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement

Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit.

The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff.

The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

59.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

59.6.1. Action items, Action Owners, Tracking, Checklist etc.

60. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is

introduced. • Identification Method • Type of Defect • Projects and Products in which problems exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References

61. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples

1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss

2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted

3 A minor incident with low impact • A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

Blocker — Highest priority. Indicates that this issue takes precedence over all others.

Critical — Indicates that this issue is causing a problem and requires urgent attention.

Major — Indicates that this issue has a significant impact.

Minor — Indicates that this issue has a relatively minor impact.

Trivial — Lowest priority.

62. Defect Fix timeline

Severity Response Time Resolution Time 1 2 3 4

63. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

64. What is A Defect?

A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

65. Defect Management Life Cycle

When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected,

postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

**66. Defect Life Cycle in Detail**

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

**66.1. Defect Workflow**

The diagram below shows the actual life cycle of a defect.

**66.2. Defect States**

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.
- cc) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- dd) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.
- ee) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.
- ff) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".
- #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".
- #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".
- #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.
- #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.
- #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.
- #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

**66.3. Guidelines for Implementing a Defect Life Cycle**

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The

defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

67. Goals of Defect Management Process (DMP) Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

68. Defect Management

68.1. Defect Tracking In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

68.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

68.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

68.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Defect Retested By
Defect Tracking	By 1	Unit Testing (UT)	DEV	DEV Team
Team 2	System Testing (ST)	ST	ST	Team
Team 3	System Integration Testing (SIT)	SIT	SIT	Team
Team 4	User Acceptance Testing (UAT)	UAT	UAT Team	Test Manager

68.3. Defect Triage Calls

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

68.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

68.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every

single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

#### 68.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

#### 68.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

##### 68.5.1. Defect Trends by Release

- o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

##### 68.5.2. Defects Density

- o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.
- o Defect Density = Total Defect/Size of the Release

##### 68.5.3. Defect Fix Rate per Release

- o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .
- o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

##### 68.5.4. Defect Aging

- o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

##### 68.5.5. Defect Analysis through RCA

- o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.
- o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.
- Steps to create a Fishbone Diagram:
  - Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
  - Follow the steps below to create a fishbone diagram:
    - (29) Write the problem at the head of the fish.
    - (30) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
    - (31) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
    - (32) Extend the causes to secondary, tertiary, and more levels as applicable.
- o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement. Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads

more effectively. • Improving productivity. • Improving profitability. Pareto Analysis Steps 1. Identify and List Problems 2. Identify the Root Cause of Each Problem 3. Score Problems 4. Group Problems Together 5. Add up Scores for Each Group 6. Take Action 68.6. Defect Prevention Processes & Improvements Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks: #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products. #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system. #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects. #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis 68.6.1. Action items, Action Owners, Tracking, Checklist etc. 69. Defect Data • Name of the Person • Types of Testing • Problem Summary • Detailed Description of Defect. • Steps to Reproduce • Life Cycle Phase • Work product where Defect was introduced. • Severity and Priority • Subsystem or Component where the Defect is introduced. • Project Activity occurs when the Defect is introduced. • Identification Method • Type of Defect • Projects and Products in which problems exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References 70. Defect Priority and Severity Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted 3 A minor incident with low impact • A minor inconvenience to customers, workaround available In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs Blocker — Highest priority. Indicates that this issue takes precedence over all others. Critical — Indicates that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority. 71. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 72. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process. 73. What is A Defect? A Defect, in

simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 74.

**Defect Management Life Cycle** When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project. 75.

**Defect Life Cycle in Detail** The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again. 75.1. Defect Workflow

The diagram below shows the actual life cycle of a defect. 75.2. Defect States #1) New:

This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle. #2) Assigned:

In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer. #3) Open:

Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason. gg)

Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer. hh) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer. ii) Deferred:

If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect

Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

1. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

2. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

3. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

3.1. Defect Workflow The diagram below shows the actual life cycle of a defect.

3.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.
- a) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- b) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.
- c) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.
- d) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".
- #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".
- #5) Pending Retest: After fixing the defect, the developer assigns the defect to the

tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest". #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

3.3.Guidelines for Implementing a Defect Life Cycle Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

4. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

5. Defect Management

5.1.Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

5.1.1.Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

5.1.2.Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

5.2.Testing stages and defects Responsibilities

SL No. Testing Stage Environment Defect Owner Defect Retested By Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team 4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager

5.3.Defect Triage Calls

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need

immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

5.3.1.Thumb rules of Triaging o All reported defects have been reviewed o All accepted defects have been prioritized o All accepted defects have severity attached o All rejected defects should have reasonable explanations for the testing team o Every defect has been assigned to an appropriate owner, individual or team o Root cause analysis for every accepted defect has been done

5.3.2.How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

5.4.Defect Metrics Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

5.5.Defect Reporting Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

5.5.1.Defect Trends by Release o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

5.5.2.Defects Density o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

5.5.3.Defect Fix Rate per Release o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

5.5.4.Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

5.5.5.Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone"

diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms. Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.

Follow the steps below to create a fishbone diagram: (1) Write the problem at the head of the fish. (2) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N] (3) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N. (4) Extend the causes to secondary, tertiary, and more levels as applicable. o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps 1. Identify and List Problems 2. Identify the Root Cause of Each Problem 3. Score Problems 4. Group Problems Together 5. Add up Scores for Each Group 6. Take Action 5.6.Defect Prevention Processes & Improvements Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks: #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products. #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system. #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects. #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis 5.6.1.Action items, Action Owners, Tracking, Checklist etc. 6. Defect Data • Name of the Person • Types of Testing • Problem Summary • Detailed Description of Defect. • Steps to Reproduce • Life Cycle Phase • Work product where Defect was introduced. • Severity and Priority • Subsystem or Component where the Defect is introduced. • Project Activity occurs when the Defect is introduced. • Identification Method • Type of Defect • Projects and Products in which problems exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References 7. Defect Priority and Severity Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is

breached • Customer data loss 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted 3 A minor incident with low impact • A minor inconvenience to customers, workaround available In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs Blocker — Highest priority. Indicates that this issue takes precedence over all others. Critical — Indicates that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority. 8. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 9. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process. 10. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 11. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project. 12. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again. 12.1. Defect Workflow The diagram below shows

the actual life cycle of a defect.

12.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.
- e) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- f) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.
- g) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.
- h) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".
- #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".
- #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".
- #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.
- #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.
- #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.
- #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

12.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

13. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

14. Defect Management

14.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier.

Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

14.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction. o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects. o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

14.1.2. Why is Defect Tracking Necessary? o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process. o Monitors the bugs that had already been resolved, thereby saving time. o To ensure that right defects are being worked on.

14.2. Testing stages and defects Responsibilities SL No. Testing Stage Environment Defect Owner Defect Retested By Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team 4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager

14.3. Defect Triage Calls Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

14.3.1. Thumb rules of Triaging o All reported defects have been reviewed o All accepted defects have been prioritized o All accepted defects have severity attached o All rejected defects should have reasonable explanations for the testing team o Every defect has been assigned to an appropriate owner, individual or team o Root cause analysis for every accepted defect has been done

14.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

14.4. Defect Metrics Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

14.5. Defect Reporting Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management

board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

14.5.1. Defect Trends by Release o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

14.5.2. Defects Density o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

14.5.3. Defect Fix Rate per Release o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

14.5.4. Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

14.5.5. Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish. Follow the steps below to create a fishbone diagram:

- (5) Write the problem at the head of the fish.
- (6) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
- (7) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
- (8) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit.

The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

14.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can

be used to support analyzing defects. #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

#### 14.6.1. Action items, Action Owners, Tracking, Checklist etc.

15. Defect Data • Name of the Person • Types of Testing • Problem Summary • Detailed Description of Defect. • Steps to Reproduce • Life Cycle Phase • Work product where Defect was introduced. • Severity and Priority • Subsystem or Component where the Defect is introduced. • Project Activity occurs when the Defect is introduced. • Identification Method • Type of Defect • Projects and Products in which problems exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References

#### 16. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples

- 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss
- 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted
- 3 A minor incident with low impact • A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

#### 17. Defect Fix timeline

Severity	Response Time	Resolution Time
1	1	2
2	2	3
3	4	18.

Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

#### 19. What is A Defect?

A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

#### 20. Defect Management Life Cycle

When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a

flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

### 21. Defect Life Cycle in Detail

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

#### 21.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

#### 21.2. Defect States

#1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.

#2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.

#3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

- i) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- j) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.
- k) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.
- l) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".

#4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".

#5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".

#6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.

#7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.

#8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.

#9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

#### 21.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be

properly documented to avoid any confusion in the future. • Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results. • Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily. • The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

## 22. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

## 23. Defect Management

### 23.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

#### 23.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

#### 23.1.2. Why is Defect Tracking Necessary?

- A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- Monitors the bugs that had already been resolved, thereby saving time.
- To ensure that right defects are being worked on.

### 23.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Defect Retested By
1	Unit Testing (UT)	DEV	DEV Team	Dev Team
2	System Testing (ST)	ST	ST Team	ST Team
3	System Integration Testing (SIT)	SIT	SIT Team	SIT Team
4	User Acceptance Testing (UAT)	UAT	UAT Team	UAT Team

### 23.3. Defect Triage

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the “right” defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

#### 23.3.1. Thumb rules of Triaging

- All reported defects have been reviewed
- All accepted defects have been prioritized
- All accepted defects have severity attached
- All rejected defects should have reasonable explanations for the testing team
- Every defect has been assigned to an appropriate owner, individual or team
- Root cause analysis for every accepted defect has been done

#### 23.3.2. How does Defect Triaging work?

The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected

at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

### 23.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

### 23.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

#### 23.5.1. Defect Trends by Release

o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

#### 23.5.2. Defects Density

o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

#### 23.5.3. Defect Fix Rate per Release

o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

#### 23.5.4. Defect Aging

o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

#### 23.5.5. Defect Analysis through RCA

o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
- Follow the steps below to create a fishbone diagram:
  - (9) Write the problem at the head of the fish.
  - (10) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
  - (11) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
  - (12) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement. Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to

focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

### 23.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

### 23.6.1. Action items, Action Owners, Tracking, Checklist etc.

24. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was resolved and recommendations for testing.
- References

25. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first?

Severity Description Examples

- 1 A critical incident with very high impact
- A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss
- 2 A major incident with significant impact
- A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted
- 3 A minor incident with low impact
- A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

26. Defect Fix timeline

Severity Response Time Resolution Time

1	2	3	4
---	---	---	---

27. Conclusion

This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process

Defect management process is mainly used to improve the

efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

28. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

29. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

30. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

30.1. Defect Workflow The diagram below shows the actual life cycle of a defect.

30.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

m)

Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.

n) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.

o) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a

case, developer can change the status of the defect as 'Deferred'. p) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug". #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed". #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest". #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 30.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 31. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 32. Defect Management

#### 32.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

##### 32.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

##### 32.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

#### 32.2. Testing stages and defects Responsibilities

SL No. Testing Stage Environment Defect Owner Defect Retested By

Defect Tracking	By 1 Unit Testing (UT)	DEV DEV Team	Dev Team	2 System Testing (ST)	ST ST Team	ST Team	ST Team	3 System Integration Testing (SIT)	SIT SIT Team	SIT Team	SIT Team	4 User Acceptance Testing (UAT)	UAT UAT Team	UAT Team	Test Manager
-----------------	------------------------	--------------	----------	-----------------------	------------	---------	---------	------------------------------------	--------------	----------	----------	---------------------------------	--------------	----------	--------------

### 32.3. Defect Triage

Calls Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

32.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

32.3.2. How does Defect Triaging work?

The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

32.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

32.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

32.5.1. Defect Trends by Release

- o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

32.5.2. Defects Density

- o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.
- o 
$$\text{Defect Density} = \frac{\text{Total Defect}}{\text{Size of the Release}}$$

32.5.3. Defect Fix Rate per Release

- o 
$$\text{Defect removal efficiency} = (1 - \frac{\text{total defects caught by customer}}{\text{total number of defects}}) \times 100$$

o Residual defect density = 
$$\frac{\text{total number of defects found by a customer}}{\text{(Total number of defects including customer found defects)}}$$
 x 100.

32.5.4. Defect Aging

- o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the

defect was fixed (if the defect is already fixed). 32.5.5. Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization. o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms. Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish. Follow the steps below to create a fishbone diagram: (13)Write the problem at the head of the fish. (14)Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N] (15)Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N. (16)Extend the causes to secondary, tertiary, and more levels as applicable. o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

32.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

32.6.1. Action items, Action Owners, Tracking, Checklist etc.

33. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was resolved and recommendations for testing.
- References

34. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does

an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted 3 A minor incident with low impact • A minor inconvenience to customers, workaround available In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs Blocker — Highest priority. Indicates that this issue takes precedence over all others. Critical — Indicates that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority. 35. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 36. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process. 37. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 38. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is

closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

### 39. Defect Life Cycle in Detail

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

#### 39.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

#### 39.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.
- q) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- r) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.
- s) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.
- t) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".
- #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".
- #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".
- #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.
- #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.
- #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.
- #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 39.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 40. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 41. Defect Management

#### 41.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the

product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier.

Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

41.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

41.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

41.2. Testing stages and defects Responsibilities SL No. Testing Stage Environment Defect Owner Defect Retested By Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team 4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager

41.3. Defect Triage Calls Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

41.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team

o Every defect has been assigned to an appropriate owner, individual or team

- o Root cause analysis for every accepted defect has been done

41.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

41.4. Defect Metrics Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by

Priority • Defect distribution by type • Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

41.5. Defect Reporting Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

41.5.1. Defect Trends by Release o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

41.5.2. Defects Density o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

41.5.3. Defect Fix Rate per Release o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

41.5.4. Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

41.5.5. Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish. Follow the steps below to create a fishbone diagram:

- (17) Write the problem at the head of the fish.
- (18) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
- (19) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
- (20) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement

Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit.

The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes.

Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

41.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the

diagram below is pretty much half of our testing tasks: #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products. #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system. #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects. #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

41.6.1. Action items, Action Owners, Tracking, Checklist etc.

42. Defect Data • Name of the Person • Types of Testing • Problem Summary • Detailed Description of Defect. • Steps to Reproduce • Life Cycle Phase • Work product where Defect was introduced. • Severity and Priority • Subsystem or Component where the Defect is introduced. • Project Activity occurs when the Defect is introduced. • Identification Method • Type of Defect • Projects and Products in which problems exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References

43. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples

1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss

2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted

3 A minor incident with low impact • A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

Blocker — Highest priority. Indicates that this issue takes precedence over all others.

Critical — Indicates that this issue is causing a problem and requires urgent attention.

Major — Indicates that this issue has a significant impact.

Minor — Indicates that this issue has a relatively minor impact.

Trivial — Lowest priority.

44. Defect Fix timeline

Severity Response Time Resolution Time 1 2 3 4

45. Conclusion

This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process

Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

46. What is A Defect?

A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

47.

**Defect Management Life Cycle** When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

**48. Defect Life Cycle in Detail** The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

**48.1. Defect Workflow** The diagram below shows the actual life cycle of a defect.

**48.2. Defect States**

- #1) **New:** This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) **Assigned:** In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) **Open:** Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.
- u) **Rejected:** If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- v) **Duplicate:** If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.
- w) **Deferred:** If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.
- x) **Not a Bug:** If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".
- #4) **Fixed:** When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".
- #5) **Pending Retest:** After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".
- #6) **Retest:** At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.
- #7) **Reopen:** If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.
- #8) **Verified:** If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.
- #9) **Closed:** When the defect does not exist any

longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 48.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 49. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 50. Defect Management

#### 50.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

##### 50.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

##### 50.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

#### 50.2. Testing stages and defects Responsibilities

SL No. Testing Stage Environment Defect Owner Defect Retested By Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team 4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager

### 50.3. Defect Triage

Calls Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

#### 50.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause

analysis for every accepted defect has been done 50.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

#### 50.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

#### 50.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

##### 50.5.1. Defect Trends by Release

- o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

##### 50.5.2. Defects Density

- o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.
- o Defect Density = Total Defect/Size of the Release

##### 50.5.3. Defect Fix Rate per Release

- o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .
- o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

##### 50.5.4. Defect Aging

- o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

##### 50.5.5. Defect Analysis through RCA

- o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.
- o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
- Follow the steps below to create a fishbone diagram:
  - (21) Write the problem at the head of the fish.
  - (22) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
  - (23) Identify the primary causes under each

category and mark it as primary cause 1, primary cause 2, primary cause N. (24) Extend the causes to secondary, tertiary, and more levels as applicable.

- o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

#### 50.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

#### 50.6.1. Action items, Action Owners, Tracking, Checklist etc.

51. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was resolved and recommendations for testing.
- References

52. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first?

Severity Description Examples

- 1 A critical incident with very high impact
- A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss
- 2 A major incident with significant impact
- A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted
- 3 A minor incident with low impact
- A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

53. Defect Fix timeline

Severity	Response Time	Resolution Time	1	2	3
----------	---------------	-----------------	---	---	---

4 54. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

55. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

56. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

57. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

57.1. Defect Workflow The diagram below shows the actual life cycle of a defect.

57.2. Defect States #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.

#2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.

#3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below

namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

- y) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- z) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.
- aa) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.
- bb) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".

#4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".

#5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".

#6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.

#7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.

#8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.

#9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 57.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 58. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 59. Defect Management

#### 59.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

#### 59.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

#### 59.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is

missed for evaluation process.

- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

59.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Defect Retested By
Defect Tracking	By 1 Unit Testing (UT)	DEV	DEV Team	Dev Team
Team 2	System Testing (ST)	ST	ST	ST
Team ST	Team ST	Team 3	System Integration Testing (SIT)	SIT
Team SIT	Team SIT	Team SIT	Team SIT	Team SIT
Team 4	User Acceptance Testing (UAT)	UAT	UAT Team	UAT Team
	Test Manager			

59.3. Defect Triage Calls

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

59.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

59.3.2. How does Defect Triaging work?

The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

59.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

59.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

59.5.1. Defect Trends by Release

- o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

59.5.2. Defects Density

- o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of

development, divided by the size of the software.

- o Defect Density = Total Defect/Size of the Release
- 59.5.3. Defect Fix Rate per Release
- o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .
- o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

59.5.4. Defect Aging

- o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

59.5.5. Defect Analysis through RCA

- o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.
- o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
- Follow the steps below to create a fishbone diagram:
- (25) Write the problem at the head of the fish.
- (26) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
- (27) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
- (28) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement

Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit.

The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes.

Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

59.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects.

Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

59.6.1. Action items, Action Owners, Tracking, Checklist etc.

60. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems

exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References

61. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples

1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss

2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted

3 A minor incident with low impact • A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

Blocker — Highest priority. Indicates that this issue takes precedence over all others.

Critical — Indicates that this issue is causing a problem and requires urgent attention.

Major — Indicates that this issue has a significant impact.

Minor — Indicates that this issue has a relatively minor impact.

Trivial — Lowest priority.

62. Defect Fix timeline

Severity	Response Time	Resolution Time
1	1	2
2	2	3
3	4	6

3. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process

Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

64. What is A Defect?

A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

65. Defect Management Life Cycle

When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get

increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

**66. Defect Life Cycle in Detail**

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

**66.1. Defect Workflow**

The diagram below shows the actual life cycle of a defect.

**66.2. Defect States #1)** New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.

#2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.

#3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

cc) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.

dd) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.

ee) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.

ff) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".

#4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".

#5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".

#6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.

#7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.

#8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.

#9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

**66.3. Guidelines for Implementing a Defect Life Cycle**

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects

and thus, in the workflow of the Defect Life Cycle.

67. Goals of Defect Management Process (DMP) Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

68. Defect Management

68.1. Defect Tracking In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

68.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

68.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

68.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Defect Retested By
1	Unit Testing (UT)	DEV	DEV Team	Dev Team
2	System Testing (ST)	ST	ST Team	ST Team
3	System Integration Testing (SIT)	SIT	SIT Team	SIT Team
4	User Acceptance Testing (UAT)	UAT	UAT Team	UAT Team
			Test Manager	

68.3. Defect Triage

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

68.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

68.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which

ensures that chances of getting a similar defect are significantly reduced.

#### 68.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

#### 68.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

##### 68.5.1. Defect Trends by Release

o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

##### 68.5.2. Defects Density

o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

##### 68.5.3. Defect Fix Rate per Release

o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

##### 68.5.4. Defect Aging

o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

##### 68.5.5. Defect Analysis through RCA

o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
- Follow the steps below to create a fishbone diagram:
  - (29) Write the problem at the head of the fish.
  - (30) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
  - (31) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
  - (32) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement.

Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit.

The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes.

Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps 1.

Identify and List Problems 2. Identify the Root Cause of Each Problem 3. Score Problems 4. Group Problems Together 5. Add up Scores for Each Group 6. Take Action 68.6. Defect Prevention Processes & Improvements Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks: #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products. #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system. #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects. #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis 68.6.1. Action items, Action Owners, Tracking, Checklist etc. 69. Defect Data • Name of the Person • Types of Testing • Problem Summary • Detailed Description of Defect. • Steps to Reproduce • Life Cycle Phase • Work product where Defect was introduced. • Severity and Priority • Subsystem or Component where the Defect is introduced. • Project Activity occurs when the Defect is introduced. • Identification Method • Type of Defect • Projects and Products in which problems exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References 70. Defect Priority and Severity Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted 3 A minor incident with low impact • A minor inconvenience to customers, workaround available In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs Blocker — Highest priority. Indicates that this issue takes precedence over all others. Critical — Indicates that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority. 71. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 72. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process. 73. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an

application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 74.

**Defect Management Life Cycle** When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project. 75.

**Defect Life Cycle in Detail** The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again. 75.1. **Defect Workflow** The diagram below shows the actual life cycle of a defect. 75.2.

**Defect States #1) New:** This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle. #2) Assigned:

In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer. #3) Open:

Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason. gg)

**Rejected:** If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer. hh) **Duplicate:** If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer. ii) **Deferred:** If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.

**Defect Management Process** Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a

Defect Management Process.

1. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.
2. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.
3. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

- 3.1. Defect Workflow The diagram below shows the actual life cycle of a defect.
- 3.2. Defect States #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.
- a) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- b) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.
- c) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.
- d) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".
- #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".
- #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the

state of the defect remains in "Pending Retest". #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 3.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 4. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 5. Defect Management

#### 5.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

##### 5.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

##### 5.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

#### 5.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Defect Retested By	Defect Tracking By	1 Unit Testing (UT)	DEV Team	Dev Team	2 System Testing (ST)	ST Team	ST Team	ST Team	ST Team	3 System Integration Testing (SIT)	SIT Team	SIT Team	SIT Team	SIT Team	4 User Acceptance Testing (UAT)	UAT Team	UAT Team	Test Manager	

### 5.3. Defect Triage Calls

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing

a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

5.3.1.Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

5.3.2.How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

5.4.Defect Metrics Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

5.5.Defect Reporting Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

5.5.1.Defect Trends by Release

- o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

5.5.2.Defects Density

- o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.
- o Defect Density = Total Defect/Size of the Release

5.5.3.Defect Fix Rate per Release

- o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .
- o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

5.5.4.Defect Aging

- o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

5.5.5.Defect Analysis through RCA

- o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.
- o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting

ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms. Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.

Follow the steps below to create a fishbone diagram: (1) Write the problem at the head of the fish. (2) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N] (3) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N. (4) Extend the causes to secondary, tertiary, and more levels as applicable. o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps 1. Identify and List Problems 2. Identify the Root Cause of Each Problem 3. Score Problems 4.

Group Problems Together 5. Add up Scores for Each Group 6. Take Action 5.6.Defect

Prevention Processes & Improvements Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks: #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products. #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system. #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects. #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis 5.6.1.Action items, Action Owners, Tracking, Checklist etc. 6. Defect Data • Name of the Person • Types of Testing • Problem Summary • Detailed Description of Defect. • Steps to Reproduce • Life Cycle Phase • Work product where Defect was introduced. • Severity and Priority • Subsystem or Component where the Defect is introduced. • Project Activity occurs when the Defect is introduced. • Identification Method • Type of Defect • Projects and Products in which problems exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References 7. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss 2 A major incident with significant impact • A customer-facing

service is unavailable for a subset of customers • Core functionality is significantly impacted 3 A minor incident with low impact • A minor inconvenience to customers, workaround available In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs Blocker — Highest priority. Indicates that this issue takes precedence over all others. Critical — Indicates that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority. 8. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 9. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process. 10. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 11. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project. 12. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again. 12.1. Defect Workflow The diagram below shows the actual life cycle of a defect. 12.2. Defect States #1) New: This is the first state of a defect

in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle. #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer. #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason. e) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer. f) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer. g) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'. h) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug". #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed". #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest". #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 12.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 13. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 14. Defect Management

#### 14.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

#### 14.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry

as it provides a faster method of correcting the defects o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction. o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects. o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

#### 14.1.2. Why is Defect Tracking Necessary?

o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process. o Monitors the bugs that had already been resolved, thereby saving time. o To ensure that right defects are being worked on.

#### 14.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Retested By
1	Unit Testing (UT)	DEV	DEV Team	Dev Team
2	System Testing (ST)	ST	ST Team	ST Team
3	System Integration Testing (SIT)	SIT	SIT Team	SIT Team
4	User Acceptance Testing (UAT)	UAT	UAT Team	UAT Team
			Test Manager	

#### 14.3. Defect Triage

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

##### 14.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

##### 14.3.2. How does Defect Triaging work?

The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

#### 14.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

#### 14.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management

process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

14.5.1. Defect Trends by Release o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

14.5.2. Defects Density o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

14.5.3. Defect Fix Rate per Release o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

14.5.4. Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

14.5.5. Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
- Follow the steps below to create a fishbone diagram:
  - (5) Write the problem at the head of the fish.
  - (6) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
  - (7) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
  - (8) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement

Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit.

The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes.

Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

14.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects.

Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes

two major approaches: I) Pareto Analysis and II) Fishbone Analysis

#### 14.6.1. Action items,

Action Owners, Tracking, Checklist etc.

#### 15. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was resolved and recommendations for testing.
- References

#### 16. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples

- 1 A critical incident with very high impact
- A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss
- 2 A major incident with significant impact
- A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted
- 3 A minor incident with low impact
- A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

#### 17. Defect Fix timeline

Severity	Response Time	Resolution Time
1	1	2
2	2	3
3	4	18

#### 18. Conclusion

This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

#### 19. What is A Defect?

A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

#### 20. Defect Management Life Cycle

When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the

projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

### 21. Defect Life Cycle in Detail

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

#### 21.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

#### 21.2. Defect States

#1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.

#2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.

#3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

- i) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- j) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.
- k) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.
- l) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".

#4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".

#5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".

#6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.

#7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.

#8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.

#9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

#### 21.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual

who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.

- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

## 22. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

## 23. Defect Management

### 23.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

#### 23.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

#### 23.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

## 23.2. Testing stages and defects Responsibilities

SL No. Testing Stage Environment Defect Owner Defect Retested By Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team 4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager

## 23.3. Defect Triage

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the “right” defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

### 23.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

### 23.3.2. How does Defect Triaging work?

The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be

considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

### 23.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

### 23.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

#### 23.5.1. Defect Trends by Release

o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

#### 23.5.2. Defects Density

o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

#### 23.5.3. Defect Fix Rate per Release

o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

#### 23.5.4. Defect Aging

o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

#### 23.5.5. Defect Analysis through RCA

o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
- Follow the steps below to create a fishbone diagram:
  - (9) Write the problem at the head of the fish.
  - (10) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
  - (11) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
  - (12) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement

Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the

Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

### 23.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

#### 23.6.1. Action items, Action Owners, Tracking, Checklist etc.

24. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was resolved and recommendations for testing.
- References

25. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first?

Severity Description Examples

- 1 A critical incident with very high impact
- 2 A customer-facing service, like Jira Cloud, is down for all customers
- 3 Confidentiality or privacy is breached
- 4 Customer data loss

2 A major incident with significant impact

3 A customer-facing service is unavailable for a subset of customers

4 Core functionality is significantly impacted

5 A minor incident with low impact

6 A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

26. Defect Fix timeline

Severity Response Time Resolution Time

1	2	3
4	27. Conclusion	

This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in

software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

28. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

29.

Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

30. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

30.1. Defect Workflow The diagram below shows the actual life cycle of a defect.

30.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.
- m) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- n) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.
- o) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.
- p) Not a Bug: If the defect

does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug". #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed". #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest". #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 30.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 31. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 32. Defect Management

#### 32.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

##### 32.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

##### 32.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

#### 32.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Defect Retested By
1	Unit Testing (UT)	DEV	DEV Team	Dev Team
2	System Testing (ST)	ST	ST Team	ST Team
3	System Integration Testing (SIT)	SIT	SIT Team	SIT Team
4	User Acceptance Testing (UAT)	UAT	UAT Team	UAT Team

#### 32.3. Defect Triage

Defect triage is a process to prioritize defects based on severity, risk, frequency of

occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the “right” defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

32.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

32.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

32.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

32.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

32.5.1. Defect Trends by Release

- o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

32.5.2. Defects Density

- o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.
- o Defect Density = Total Defect/Size of the Release

32.5.3. Defect Fix Rate per Release

- o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .
- o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

32.5.4. Defect Aging

- o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

32.5.5. Defect Analysis through RCA

- o RCA

(Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

- o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish. Follow the steps below to create a fishbone diagram:

- (13) Write the problem at the head of the fish.
- (14) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
- (15) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
- (16) Extend the causes to secondary, tertiary, and more levels as applicable.

- o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement. Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

### 32.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

#### 32.6.1. Action items, Action Owners, Tracking, Checklist etc.

33. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was resolved and recommendations for testing.
- References

34. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing

a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted 3 A minor incident with low impact • A minor inconvenience to customers, workaround available In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs Blocker — Highest priority. Indicates that this issue takes precedence over all others. Critical — Indicates that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority. 35. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 36. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process. 37. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 38. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The

number of states that a defect goes through varies from project to project.

### 39. Defect Life Cycle in Detail

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

#### 39.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

#### 39.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.
- q) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- r) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.
- s) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.
- t) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".
- #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".
- #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".
- #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.
- #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.
- #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.
- #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

#### 39.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 40. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

#### 41. Defect Management

##### 41.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and

business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

41.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

41.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

41.2. Testing stages and defects Responsibilities SL No. Testing Stage Environment Defect Owner Defect Retested By Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team

4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager 41.3. Defect Triage Calls Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence.

Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

41.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

41.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

41.4. Defect Metrics Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA,

UAT or End-user 41.5. Defect Reporting Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

41.5.1. Defect Trends by Release o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

41.5.2. Defects Density o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

41.5.3. Defect Fix Rate per Release o Defect removal efficiency =  $(1 - (\text{total defects caught by customer/ total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

41.5.4. Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

41.5.5. Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish. Follow the steps below to create a fishbone diagram:

- (17) Write the problem at the head of the fish.
- (18) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
- (19) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
- (20) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

41.6. Defect Prevention Processes & Improvements Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

#1) Review and Inspection: This

method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products. #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system. #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects. #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

41.6.1. Action items, Action Owners, Tracking, Checklist etc.

42. Defect Data • Name of the Person • Types of Testing • Problem Summary • Detailed Description of Defect. • Steps to Reproduce • Life Cycle Phase • Work product where Defect was introduced. • Severity and Priority • Subsystem or Component where the Defect is introduced. • Project Activity occurs when the Defect is introduced. • Identification Method • Type of Defect • Projects and Products in which problems exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References

43. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples

- 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss
- 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted
- 3 A minor incident with low impact • A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

44. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 5. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

46. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

47. Defect Management Life Cycle When a system gives a different output other than the actual

business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

**48. Defect Life Cycle in Detail**

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

**48.1. Defect Workflow**

The diagram below shows the actual life cycle of a defect.

**48.2. Defect States #1)** **New:** This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.

**#2) Assigned:** In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.

**#3) Open:** Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

**u) Rejected:** If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.

**v) Duplicate:** If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.

**w) Deferred:** If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.

**x) Not a Bug:** If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".

**#4) Fixed:** When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".

**#5) Pending Retest:** After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".

**#6) Retest:** At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.

**#7) Reopen:** If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.

**#8) Verified:** If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.

**#9) Closed:** When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields

where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter. 48.3. Guidelines for Implementing a Defect Life Cycle Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

49. Goals of Defect Management Process (DMP) Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

50. Defect Management 50.1. Defect Tracking In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

50.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

50.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

50.2. Testing stages and defects Responsibilities SL No. Testing Stage Environment Defect Owner Defect Retested By Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team

4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager 50.3. Defect Triage Calls Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

50.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

50.3.2. How does Defect Triaging work? The

defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

#### 50.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

#### 50.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

##### 50.5.1. Defect Trends by Release

o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

##### 50.5.2. Defects Density

o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

##### 50.5.3. Defect Fix Rate per Release

o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

##### 50.5.4. Defect Aging

o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

##### 50.5.5. Defect Analysis through RCA

o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
- Follow the steps below to create a fishbone diagram:
  - (21) Write the problem at the head of the fish.
  - (22) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
  - (23) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
  - (24) Extend the

causes to secondary, tertiary, and more levels as applicable.

- o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement. Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

#### 50.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

#### 50.6.1. Action items, Action Owners, Tracking, Checklist etc.

51. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was resolved and recommendations for testing.
- References

52. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples

- 1 A critical incident with very high impact
- A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss
- 2 A major incident with significant impact
- A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted
- 3 A minor incident with low impact
- A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

53. Defect Fix timeline

Severity Response Time Resolution Time

- 1
- 2
- 3
- 4
54. Conclusion

This was all there is to know about the Bug Life Cycle and how the testing

and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

55. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

56. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

57. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

57.1. Defect Workflow The diagram below shows the actual life cycle of a defect.

57.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.
- y) Rejected:

If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer. z) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer. aa) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'. bb) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug". #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed". #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest". #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

57.3. Guidelines for Implementing a Defect Life Cycle Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

58. Goals of Defect Management Process (DMP) Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

59. Defect Management

59.1. Defect Tracking In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

59.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

59.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby

saving time. o To ensure that right defects are being worked on. 59.2. Testing stages and defects Responsibilities SL No. Testing Stage Environment Defect Owner Defect Retested By Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team 4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager 59.3. Defect Triage Calls Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test. 59.3.1. Thumb rules of Triaging o All reported defects have been reviewed o All accepted defects have been prioritized o All accepted defects have severity attached o All rejected defects should have reasonable explanations for the testing team o Every defect has been assigned to an appropriate owner, individual or team o Root cause analysis for every accepted defect has been done 59.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced. 59.4. Defect Metrics Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

59.5. Defect Reporting Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them. 59.5.1. Defect Trends by Release o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed. 59.5.2. Defects Density o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the

Release 59.5.3. Defect Fix Rate per Release o Defect removal efficiency = (1 - (total defects caught by customer/ total number of defects)) x 100. o Residual defect density = (total number of defects found by a customer)/ (Total number of defects including customer found defects) x 100. 59.5.4. Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed). 59.5.5. Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization. o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms. Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish. Follow the steps below to create a fishbone diagram: (25)Write the problem at the head of the fish. (26)Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N] (27)Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N. (28)Extend the causes to secondary, tertiary, and more levels as applicable. o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

59.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

59.6.1. Action items, Action Owners, Tracking, Checklist etc.

60. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.

**Impact on Project** • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References 61. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted 3 A minor incident with low impact • A minor inconvenience to customers, workaround available In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs Blocker — Highest priority. Indicates that this issue takes precedence over all others. Critical — Indicates that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority.

**62. Defect Fix timeline Severity Response Time Resolution Time**

1	2	3
---	---	---

**63. Conclusion** This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

**64. What is A Defect?** A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

**65. Defect Management Life Cycle** When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In

order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

### 66. Defect Life Cycle in Detail

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

#### 66.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

#### 66.2. Defect States

#1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.

#2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.

#3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

cc) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.

dd) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.

ee) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.

ff) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".

#4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".

#5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".

#6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.

#7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.

#8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.

#9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

#### 66.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 67. Goals of Defect Management Process

(DMP) Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

68. Defect Management 68.1. Defect Tracking In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

68.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

68.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

68.2. Testing stages and defects Responsibilities SL No. Testing Stage Environment Defect Owner Defect Retested By Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team

4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager 68.3. Defect Triage Calls Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

68.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

68.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

68.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

68.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

68.5.1. Defect Trends by Release

o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

68.5.2. Defects Density

o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

68.5.3. Defect Fix Rate per Release

o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

68.5.4. Defect Aging

o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

68.5.5. Defect Analysis through RCA

o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram

A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
- Follow the steps below to create a fishbone diagram:
  - (29) Write the problem at the head of the fish.
  - (30) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
  - (31) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
  - (32) Extend the causes to secondary, tertiary, and more levels as applicable.
- o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement.

Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit.

The 80/20 Rule

Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes.

Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
- 4.

Group Problems Together 5. Add up Scores for Each Group 6. Take Action 68.6. Defect Prevention Processes & Improvements Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks: #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products. #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system. #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects. #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis 68.6.1. Action items, Action Owners, Tracking, Checklist etc. 69. Defect Data • Name of the Person • Types of Testing • Problem Summary • Detailed Description of Defect. • Steps to Reproduce • Life Cycle Phase • Work product where Defect was introduced. • Severity and Priority • Subsystem or Component where the Defect is introduced. • Project Activity occurs when the Defect is introduced. • Identification Method • Type of Defect • Projects and Products in which problems exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References 70. Defect Priority and Severity Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted 3 A minor incident with low impact • A minor inconvenience to customers, workaround available In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs Blocker — Highest priority. Indicates that this issue takes precedence over all others. Critical — Indicates that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority. 71. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 72. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process. 73. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The

defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 74.

**Defect Management Life Cycle** When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project. 75. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again. 75.1. Defect Workflow The diagram below shows the actual life cycle of a defect. 75.2. Defect States #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle. #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer. #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason. gg) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer. hh) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer. ii) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.

**Defect Management Process** Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The

organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

1. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.
2. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.
3. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

- 3.1. Defect Workflow The diagram below shows the actual life cycle of a defect.
- 3.2. Defect States #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.
- a) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- b) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.
- c) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.
- d) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".
- #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".
- #5) Pending Retest: After fixing the defect, the developer assigns the defect to the

tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest". #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

3.3.Guidelines for Implementing a Defect Life Cycle Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

4. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

5. Defect Management

5.1.Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

5.1.1.Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

5.1.2.Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

5.2.Testing stages and defects Responsibilities

SL No. Testing Stage Environment Defect Owner Defect Retested By Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team 4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager

5.3.Defect Triage Calls

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need

immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

5.3.1.Thumb rules of Triaging o All reported defects have been reviewed o All accepted defects have been prioritized o All accepted defects have severity attached o All rejected defects should have reasonable explanations for the testing team o Every defect has been assigned to an appropriate owner, individual or team o Root cause analysis for every accepted defect has been done

5.3.2.How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

5.4.Defect Metrics Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

5.5.Defect Reporting Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

5.5.1.Defect Trends by Release o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

5.5.2.Defects Density o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

5.5.3.Defect Fix Rate per Release o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

5.5.4.Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

5.5.5.Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone"

diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms. Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.

Follow the steps below to create a fishbone diagram: (1) Write the problem at the head of the fish. (2) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N] (3) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N. (4) Extend the causes to secondary, tertiary, and more levels as applicable. o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps 1. Identify and List Problems 2. Identify the Root Cause of Each Problem 3. Score Problems 4. Group Problems Together 5. Add up Scores for Each Group 6. Take Action 5.6.Defect Prevention Processes & Improvements Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks: #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products. #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system. #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects. #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis 5.6.1.Action items, Action Owners, Tracking, Checklist etc. 6. Defect Data • Name of the Person • Types of Testing • Problem Summary • Detailed Description of Defect. • Steps to Reproduce • Life Cycle Phase • Work product where Defect was introduced. • Severity and Priority • Subsystem or Component where the Defect is introduced. • Project Activity occurs when the Defect is introduced. • Identification Method • Type of Defect • Projects and Products in which problems exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References 7. Defect Priority and Severity Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is

breached • Customer data loss 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted 3 A minor incident with low impact • A minor inconvenience to customers, workaround available In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs Blocker — Highest priority. Indicates that this issue takes precedence over all others. Critical — Indicates that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority. 8. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 9. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process. 10. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 11. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project. 12. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again. 12.1. Defect Workflow The diagram below shows

the actual life cycle of a defect.

12.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.
- e) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- f) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.
- g) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.
- h) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".
- #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".
- #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".
- #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.
- #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.
- #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.
- #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

12.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

13. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

14. Defect Management

14.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier.

Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

14.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction. o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects. o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

14.1.2. Why is Defect Tracking Necessary? o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process. o Monitors the bugs that had already been resolved, thereby saving time. o To ensure that right defects are being worked on.

14.2. Testing stages and defects Responsibilities SL No. Testing Stage Environment Defect Owner Defect Retested By Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team 4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager

14.3. Defect Triage Calls Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

14.3.1. Thumb rules of Triaging o All reported defects have been reviewed o All accepted defects have been prioritized o All accepted defects have severity attached o All rejected defects should have reasonable explanations for the testing team o Every defect has been assigned to an appropriate owner, individual or team o Root cause analysis for every accepted defect has been done

14.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

14.4. Defect Metrics Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

14.5. Defect Reporting Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management

board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

14.5.1. Defect Trends by Release o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

14.5.2. Defects Density o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

14.5.3. Defect Fix Rate per Release o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

14.5.4. Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

14.5.5. Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish. Follow the steps below to create a fishbone diagram:

- (5) Write the problem at the head of the fish.
- (6) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
- (7) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
- (8) Extend the causes to secondary, tertiary, and more levels as applicable.

o Pareto charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

14.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can

be used to support analyzing defects. #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

#### 14.6.1. Action items, Action Owners, Tracking, Checklist etc.

15. Defect Data • Name of the Person • Types of Testing • Problem Summary • Detailed Description of Defect. • Steps to Reproduce • Life Cycle Phase • Work product where Defect was introduced. • Severity and Priority • Subsystem or Component where the Defect is introduced. • Project Activity occurs when the Defect is introduced. • Identification Method • Type of Defect • Projects and Products in which problems exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References

#### 16. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples

- 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss
- 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted
- 3 A minor incident with low impact • A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

#### 17. Defect Fix timeline

Severity Response Time Resolution Time 1 2 3 4

#### 18. Conclusion

This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

#### 19. What is A Defect?

A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

#### 20. Defect Management Life Cycle

When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a

flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

### 21. Defect Life Cycle in Detail

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

#### 21.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

#### 21.2. Defect States

#1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.

#2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.

#3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

- i) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- j) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.
- k) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.
- l) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".

#4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".

#5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".

#6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.

#7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.

#8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.

#9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

#### 21.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be

properly documented to avoid any confusion in the future. • Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results. • Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily. • The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

## 22. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

## 23. Defect Management

### 23.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

#### 23.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

#### 23.1.2. Why is Defect Tracking Necessary?

- A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- Monitors the bugs that had already been resolved, thereby saving time.
- To ensure that right defects are being worked on.

#### 23.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Defect Retested By
1	Unit Testing (UT)	DEV	DEV Team	Dev Team
2	System Testing (ST)	ST	ST Team	ST Team
3	System Integration Testing (SIT)	SIT	SIT Team	SIT Team
4	User Acceptance Testing (UAT)	UAT	UAT Team	UAT Team

#### 23.3. Defect Triage

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the “right” defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

##### 23.3.1. Thumb rules of Triaging

- All reported defects have been reviewed
- All accepted defects have been prioritized
- All accepted defects have severity attached
- All rejected defects should have reasonable explanations for the testing team
- Every defect has been assigned to an appropriate owner, individual or team
- Root cause analysis for every accepted defect has been done

##### 23.3.2. How does Defect Triaging work?

The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected

at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

### 23.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

### 23.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

#### 23.5.1. Defect Trends by Release

o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

#### 23.5.2. Defects Density

o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

#### 23.5.3. Defect Fix Rate per Release

o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

#### 23.5.4. Defect Aging

o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

#### 23.5.5. Defect Analysis through RCA

o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
- Follow the steps below to create a fishbone diagram:
  - (9) Write the problem at the head of the fish.
  - (10) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
  - (11) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
  - (12) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement

Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to

focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

23.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

23.6.1. Action items, Action Owners, Tracking, Checklist etc.

24. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was resolved and recommendations for testing.
- References

25. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first?

Severity Description Examples

- 1 A critical incident with very high impact
- A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss
- 2 A major incident with significant impact
- A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted
- 3 A minor incident with low impact
- A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

26. Defect Fix timeline

Severity Response Time Resolution Time

1	2	3	4	27. Conclusion
---	---	---	---	----------------

This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process

Defect management process is mainly used to improve the

efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

28. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

29. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

30. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

30.1. Defect Workflow The diagram below shows the actual life cycle of a defect.

30.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

m)

Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.

n) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.

o) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a

case, developer can change the status of the defect as 'Deferred'. p) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug". #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed". #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest". #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 30.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 31. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 32. Defect Management

#### 32.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

##### 32.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

##### 32.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

#### 32.2. Testing stages and defects Responsibilities

SL No. Testing Stage Environment Defect Owner Defect Retested By

Defect Tracking	By 1 Unit Testing (UT)	DEV DEV Team	Dev Team	2 System Testing (ST)	ST ST Team	ST Team	ST Team	ST Team	3 System Integration Testing (SIT)	SIT SIT Team	SIT Team	SIT Team	4 User Acceptance Testing (UAT)	UAT UAT Team	UAT Team	Test Manager
-----------------	------------------------	--------------	----------	-----------------------	------------	---------	---------	---------	------------------------------------	--------------	----------	----------	---------------------------------	--------------	----------	--------------

#### 32.3. Defect Triage

Calls Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the “right” defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

32.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

32.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

32.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

32.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

32.5.1. Defect Trends by Release

- o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

32.5.2. Defects Density

- o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.
- o 
$$\text{Defect Density} = \frac{\text{Total Defect}}{\text{Size of the Release}}$$

32.5.3. Defect Fix Rate per Release

- o 
$$\text{Defect removal efficiency} = (1 - \frac{\text{total defects caught by customer}}{\text{total number of defects}}) \times 100$$

o Residual defect density = 
$$\frac{\text{total number of defects found by a customer}}{\text{(Total number of defects including customer found defects)}}$$
 x 100.

32.5.4. Defect Aging

- o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the

defect was fixed (if the defect is already fixed). 32.5.5. Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization. o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms. Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish. Follow the steps below to create a fishbone diagram: (13)Write the problem at the head of the fish. (14)Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N] (15)Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N. (16)Extend the causes to secondary, tertiary, and more levels as applicable. o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

32.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

32.6.1. Action items, Action Owners, Tracking, Checklist etc.

33. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was resolved and recommendations for testing.
- References

34. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does

an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted 3 A minor incident with low impact • A minor inconvenience to customers, workaround available In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs Blocker — Highest priority. Indicates that this issue takes precedence over all others. Critical — Indicates that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority. 35. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 36. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process. 37. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 38. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is

closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

### 39. Defect Life Cycle in Detail

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

#### 39.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

#### 39.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.
- q) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- r) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.
- s) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.
- t) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".
- #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".
- #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".
- #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.
- #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.
- #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.
- #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 39.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 40. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 41. Defect Management

#### 41.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the

product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier.

Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

41.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

41.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

41.2. Testing stages and defects Responsibilities SL No. Testing Stage Environment Defect Owner Defect Retested By Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team 4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager

41.3. Defect Triage Calls Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

41.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team

o Every defect has been assigned to an appropriate owner, individual or team

- o Root cause analysis for every accepted defect has been done

41.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

41.4. Defect Metrics Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by

Priority • Defect distribution by type • Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

41.5. Defect Reporting Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

41.5.1. Defect Trends by Release o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

41.5.2. Defects Density o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

41.5.3. Defect Fix Rate per Release o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

41.5.4. Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

41.5.5. Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish. Follow the steps below to create a fishbone diagram:

- (17) Write the problem at the head of the fish.
- (18) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
- (19) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
- (20) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement

Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit.

The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes.

Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

41.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the

diagram below is pretty much half of our testing tasks: #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products. #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system. #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects. #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

41.6.1. Action items, Action Owners, Tracking, Checklist etc.

42. Defect Data • Name of the Person • Types of Testing • Problem Summary • Detailed Description of Defect. • Steps to Reproduce • Life Cycle Phase • Work product where Defect was introduced. • Severity and Priority • Subsystem or Component where the Defect is introduced. • Project Activity occurs when the Defect is introduced. • Identification Method • Type of Defect • Projects and Products in which problems exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References

43. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples

1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss

2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted

3 A minor incident with low impact • A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

Blocker — Highest priority. Indicates that this issue takes precedence over all others.

Critical — Indicates that this issue is causing a problem and requires urgent attention.

Major — Indicates that this issue has a significant impact.

Minor — Indicates that this issue has a relatively minor impact.

Trivial — Lowest priority.

44. Defect Fix timeline

Severity Response Time Resolution Time 1 2 3 4

45. Conclusion

This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process

Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

46. What is A Defect?

A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

47.

**Defect Management Life Cycle** When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

**48. Defect Life Cycle in Detail** The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

**48.1. Defect Workflow** The diagram below shows the actual life cycle of a defect.

**48.2. Defect States**

- #1) **New:** This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) **Assigned:** In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) **Open:** Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.
- u) **Rejected:** If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- v) **Duplicate:** If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.
- w) **Deferred:** If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.
- x) **Not a Bug:** If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".
- #4) **Fixed:** When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".
- #5) **Pending Retest:** After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".
- #6) **Retest:** At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.
- #7) **Reopen:** If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.
- #8) **Verified:** If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.
- #9) **Closed:** When the defect does not exist any

longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 48.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 49. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 50. Defect Management

#### 50.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

##### 50.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

##### 50.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

#### 50.2. Testing stages and defects Responsibilities

SL No. Testing Stage Environment Defect Owner Defect Retested By

Defect Tracking	By 1 Unit Testing (UT)	DEV DEV Team	Dev Team 2 System Testing (ST)	ST ST Team	ST Team ST Team	ST Team 3 System Integration Testing (SIT)	SIT SIT Team	SIT Team SIT Team	4 User Acceptance Testing (UAT)	UAT UAT Team	UAT Team Test Manager
-----------------	------------------------	--------------	--------------------------------	------------	-----------------	--	--------------	-------------------	---------------------------------	--------------	-----------------------

#### 50.3. Defect Triage

Calls

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

##### 50.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause

analysis for every accepted defect has been done 50.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

#### 50.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

#### 50.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

##### 50.5.1. Defect Trends by Release

o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

##### 50.5.2. Defects Density

o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

##### 50.5.3. Defect Fix Rate per Release

o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

##### 50.5.4. Defect Aging

o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

##### 50.5.5. Defect Analysis through RCA

o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
- Follow the steps below to create a fishbone diagram:
  - (21) Write the problem at the head of the fish.
  - (22) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
  - (23) Identify the primary causes under each

category and mark it as primary cause 1, primary cause 2, primary cause N. (24) Extend the causes to secondary, tertiary, and more levels as applicable.

- o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

#### 50.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

#### 50.6.1. Action items, Action Owners, Tracking, Checklist etc.

51. Defect Data
  - Name of the Person
  - Types of Testing
  - Problem Summary
  - Detailed Description of Defect
  - Steps to Reproduce
  - Life Cycle Phase
  - Work product where Defect was introduced
  - Severity and Priority
  - Subsystem or Component where the Defect is introduced
  - Project Activity occurs when the Defect is introduced
  - Identification Method
  - Type of Defect
  - Projects and Products in which problems exist
  - Current Owner
  - Current State of the Report
  - Work product where Defect occurred
  - Impact on Project
  - Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect
  - Dates when various defect lifecycle phases occur
  - Description of how the defect was resolved and recommendations for testing
  - References
52. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first?

Severity Description Examples

- 1 A critical incident with very high impact
- A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss
- 2 A major incident with significant impact
- A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted
- 3 A minor incident with low impact
- A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

53. Defect Fix timeline
- Severity Response Time Resolution Time
- 1 2 3

4 54. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process. 55. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 56. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project. 57. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again. 57.1. Defect Workflow The diagram below shows the actual life cycle of a defect. 57.2. Defect States #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle. #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer. #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below

namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

- y) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- z) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.
- aa) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.
- bb) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".

#4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".

#5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".

#6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.

#7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.

#8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.

#9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 57.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 58. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 59. Defect Management

#### 59.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

#### 59.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

#### 59.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is

missed for evaluation process.

- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

59.2. Testing stages and defects Responsibilities SL No. Testing Stage Environment Defect Owner Defect Retested By

Defect Tracking	By 1 Unit Testing (UT)	DEV DEV Team	Dev Team 2 System Testing (ST)	ST ST
Team ST	Team ST	Team 3 System Integration Testing (SIT)	SIT SIT Team	SIT Team SIT Team
4 User Acceptance Testing (UAT)	UAT UAT Team	UAT Team Test Manager	59.3. Defect Triage	

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

59.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

59.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

59.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

59.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

59.5.1. Defect Trends by Release

- o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

59.5.2. Defects Density

- o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of

development, divided by the size of the software.

- o Defect Density = Total Defect/Size of the Release
- 59.5.3. Defect Fix Rate per Release
- o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .
- o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

59.5.4. Defect Aging

- o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

59.5.5. Defect Analysis through RCA

- o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.
- o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
- Follow the steps below to create a fishbone diagram:
- (25) Write the problem at the head of the fish.
- (26) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
- (27) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
- (28) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement

Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit.

The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes.

Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

59.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects.

Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

59.6.1. Action items, Action Owners, Tracking, Checklist etc.

60. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems

exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References

61. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples

1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss

2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted

3 A minor incident with low impact • A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

Blocker — Highest priority. Indicates that this issue takes precedence over all others.

Critical — Indicates that this issue is causing a problem and requires urgent attention.

Major — Indicates that this issue has a significant impact.

Minor — Indicates that this issue has a relatively minor impact.

Trivial — Lowest priority.

62. Defect Fix timeline

Severity	Response Time	Resolution Time
1	1	2
2	2	3
3	4	6

3. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process

Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

64. What is A Defect?

A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

65. Defect Management Life Cycle

When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get

increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

**66. Defect Life Cycle in Detail**

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

**66.1. Defect Workflow**

The diagram below shows the actual life cycle of a defect.

**66.2. Defect States #1)**

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

cc)

Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.

dd) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.

ee) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.

ff) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".

#4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".

#5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".

#6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.

#7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.

#8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.

#9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

**66.3. Guidelines for Implementing a Defect Life Cycle**

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects

and thus, in the workflow of the Defect Life Cycle.

67. Goals of Defect Management Process (DMP) Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

68. Defect Management

68.1. Defect Tracking In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

68.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

68.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

68.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Defect Retested By
1	Unit Testing (UT)	DEV	DEV Team	Dev Team
2	System Testing (ST)	ST	ST Team	ST Team
3	System Integration Testing (SIT)	SIT	SIT Team	SIT Team
4	User Acceptance Testing (UAT)	UAT	UAT Team	UAT Team
			Test Manager	

68.3. Defect Triage

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

68.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

68.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which

ensures that chances of getting a similar defect are significantly reduced.

#### 68.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

#### 68.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

##### 68.5.1. Defect Trends by Release

o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

##### 68.5.2. Defects Density

o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

##### 68.5.3. Defect Fix Rate per Release

o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

##### 68.5.4. Defect Aging

o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

##### 68.5.5. Defect Analysis through RCA

o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
- Follow the steps below to create a fishbone diagram:
  - (29) Write the problem at the head of the fish.
  - (30) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
  - (31) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
  - (32) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement.

Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit.

The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes.

Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps 1.

Identify and List Problems 2. Identify the Root Cause of Each Problem 3. Score Problems 4. Group Problems Together 5. Add up Scores for Each Group 6. Take Action 68.6. Defect Prevention Processes & Improvements Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks: #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products. #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system. #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects. #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis 68.6.1. Action items, Action Owners, Tracking, Checklist etc. 69. Defect Data • Name of the Person • Types of Testing • Problem Summary • Detailed Description of Defect. • Steps to Reproduce • Life Cycle Phase • Work product where Defect was introduced. • Severity and Priority • Subsystem or Component where the Defect is introduced. • Project Activity occurs when the Defect is introduced. • Identification Method • Type of Defect • Projects and Products in which problems exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References 70. Defect Priority and Severity Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted 3 A minor incident with low impact • A minor inconvenience to customers, workaround available In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs Blocker — Highest priority. Indicates that this issue takes precedence over all others. Critical — Indicates that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority. 71. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 72. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process. 73. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an

application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 74.

**Defect Management Life Cycle** When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project. 75. **Defect Life Cycle in Detail** The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again. 75.1. **Defect Workflow** The diagram below shows the actual life cycle of a defect. 75.2. **Defect States**

- #1) **New:** This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) **Assigned:** In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) **Open:** Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

gg)  
Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.  
hh) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.  
ii) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.

**Defect Management Process** Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in

software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

1. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

2. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

3. Defect Life Cycle in Detail

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

3.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

3.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

a) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.

b) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.

c) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.

d) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the

defect gets changed to "Not a Bug". #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed". #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest". #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 3.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 4. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 5. Defect Management

#### 5.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

##### 5.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

##### 5.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

#### 5.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Defect Retested By
1	Unit Testing (UT)	DEV Team	Dev Team	System Testing (ST)
2	System Testing (ST)	ST Team	ST Team	System Integration Testing (SIT)
3	SIT	SIT Team	SIT Team	SIT Team
4	User Acceptance Testing (UAT)	UAT Team	UAT Team	Test Manager

### 5.3. Defect Triage Calls

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and

tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the “right” defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

5.3.1.Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

5.3.2.How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

5.4.Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

5.5.Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

5.5.1.Defect Trends by Release

The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

5.5.2.Defects Density

o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

$$\text{o Defect Density} = \frac{\text{Total Defect}}{\text{Size of the Release}}$$

5.5.3.Defect Fix Rate per Release

o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

5.5.4.Defect Aging

o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

5.5.5.Defect Analysis through RCA

o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software

Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization. o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.

Follow the steps below to create a fishbone diagram: (1) Write the problem at the head of the fish. (2) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N] (3) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N. (4) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
  - Helping people to organize their workloads more effectively.
  - Improving productivity.
  - Improving profitability.
- Pareto Analysis Steps 1. Identify and List Problems 2. Identify the Root Cause of Each Problem 3. Score Problems 4. Group Problems Together 5. Add up Scores for Each Group 6. Take Action 5.6.Defect Prevention Processes & Improvements Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

#1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products. #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system. #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects. #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis 5.6.1.Action items, Action Owners, Tracking, Checklist etc. 6. Defect Data • Name of the Person • Types of Testing • Problem Summary • Detailed Description of Defect. • Steps to Reproduce • Life Cycle Phase • Work product where Defect was introduced. • Severity and Priority • Subsystem or Component where the Defect is introduced. • Project Activity occurs when the Defect is introduced. • Identification Method • Type of Defect • Projects and Products in which problems exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References 7. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand,

is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted 3 A minor incident with low impact • A minor inconvenience to customers, workaround available In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs Blocker — Highest priority. Indicates that this issue takes precedence over all others. Critical — Indicates that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority. 8. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 9. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process. 10. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 11. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project. 12. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects

from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

### 12.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

### 12.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.
- e) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- f) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.
- g) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.
- h) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".
- #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".
- #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".
- #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.
- #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.
- #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.
- #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 12.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 13. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 14. Defect Management

#### 14.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of

time and to effectively manage them, defect tracking system is used to make the job easier.

Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

14.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

14.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

14.2. Testing stages and defects Responsibilities SL No. Testing Stage Environment Defect Owner Defect Retested By Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team 4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager

14.3. Defect Triage Calls Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

14.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

14.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

14.4. Defect Metrics Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

14.5. Defect Reporting Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for

feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

14.5.1. Defect Trends by Release o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

14.5.2. Defects Density o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

14.5.3. Defect Fix Rate per Release o Defect removal efficiency =  $(1 - (\text{total defects caught by customer/ total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

14.5.4. Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

14.5.5. Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish. Follow the steps below to create a fishbone diagram:

- (5) Write the problem at the head of the fish.
- (6) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
- (7) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
- (8) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

14.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

#1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.

#2) Walkthrough: This is more or less like a review but it's

mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system. #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects. #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

14.6.1. Action items, Action Owners, Tracking, Checklist etc.

15. Defect Data • Name of the Person • Types of Testing • Problem Summary • Detailed Description of Defect. • Steps to Reproduce • Life Cycle Phase • Work product where Defect was introduced. • Severity and Priority • Subsystem or Component where the Defect is introduced. • Project Activity occurs when the Defect is introduced. • Identification Method • Type of Defect • Projects and Products in which problems exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References

16. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first?

Severity Description Examples

- 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss

- 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted

- 3 A minor incident with low impact • A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

17. Defect Fix timeline

Severity Response Time Resolution Time

1	2	3	4	18. Conclusion
---	---	---	---	----------------

This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process

Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

19. What is A Defect?

A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

20. Defect Management Life Cycle

When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing

team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

### 21. Defect Life Cycle in Detail

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

#### 21.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

#### 21.2. Defect States

#1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.

#2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.

#3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

- i) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- j) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.
- k) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.
- l) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".

#4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".

#5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".

#6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.

#7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.

#8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.

#9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

#### 21.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines

can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

## 22. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

## 23. Defect Management

### 23.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

#### 23.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

#### 23.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

### 23.2. Testing stages and defects Responsibilities

SL No. Testing Stage Environment Defect Owner Defect Retested By Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team 4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager

### 23.3. Defect Triage

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the “right” defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

#### 23.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

#### 23.3.2. How does Defect Triaging work?

The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and

Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

### 23.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

### 23.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

#### 23.5.1. Defect Trends by Release

- o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

#### 23.5.2. Defects Density

- o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.
- o Defect Density = Total Defect/Size of the Release

#### 23.5.3. Defect Fix Rate per Release

- o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .
- o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

#### 23.5.4. Defect Aging

- o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

#### 23.5.5. Defect Analysis through RCA

- o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.
- o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- (9) Write the problem at the head of the fish.
- (10) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
- (11) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
- (12) Extend the causes to secondary, tertiary, and more levels as applicable.

#### o Pareto charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative

impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement. Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

### 23.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

#### 23.6.1. Action items, Action Owners, Tracking, Checklist etc.

24. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was resolved and recommendations for testing.
- References

25. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first?

Severity Description Examples

- 1 A critical incident with very high impact
- A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss

- 2 A major incident with significant impact
- A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted

- 3 A minor incident with low impact
- A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

26. Defect Fix timeline
- Severity Response Time
- Resolution Time
- 1 2 3

27. Conclusion

This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software

development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

28. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

29. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

30. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

30.1. Defect Workflow The diagram below shows the actual life cycle of a defect.

30.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.
- m) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- n) Duplicate: If the developer finds the defect as same as any

other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer. o) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'. p) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug". #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed". #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest". #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 30.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 31. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 32. Defect Management

#### 32.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

##### 32.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

##### 32.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

### 32.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Retested By
1	Unit Testing	Unit Test Environment	Developer	QA Engineer
2	Integration Testing	Integration Test Environment	QA Engineer	QA Engineer
3	System Testing	System Test Environment	QA Engineer	QA Engineer
4	Acceptance Testing	Acceptance Test Environment	QA Engineer	QA Engineer
5	UAT Testing	UAT Test Environment	QA Engineer	QA Engineer

Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team 4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager 32.3. Defect Triage Calls Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the “right” defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test. 32.3.1. Thumb rules of Triaging o All reported defects have been reviewed o All accepted defects have been prioritized o All accepted defects have severity attached o All rejected defects should have reasonable explanations for the testing team o Every defect has been assigned to an appropriate owner, individual or team o Root cause analysis for every accepted defect has been done 32.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are: • The validity of the defect • Time sensitivity for resolution • The complexity involved in the resolution • Business impact This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced. 32.4. Defect Metrics Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be: • Defect distribution by cause • Defect distribution by feature/functional area • Defect distribution by Severity • Defect distribution by Priority • Defect distribution by type • Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user 32.5. Defect Reporting Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them. 32.5.1. Defect Trends by Release o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed. 32.5.2. Defects Density o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software. o Defect Density = Total Defect/Size of the Release 32.5.3. Defect Fix Rate per Release o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ . o Residual defect density = (total

number of defects found by a customer)/ (Total number of defects including customer found defects) x 100.

32.5.4. Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

32.5.5. Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish. Follow the steps below to create a fishbone diagram:

- (13) Write the problem at the head of the fish.
- (14) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
- (15) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
- (16) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement

Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit.

The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes.

Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

32.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects.

Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

32.6.1. Action items, Action Owners, Tracking, Checklist etc.

33. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was

resolved and recommendations for testing.

- References 34. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted 3 A minor incident with low impact • A minor inconvenience to customers, workaround available In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs Blocker — Highest priority. Indicates that this issue takes precedence over all others. Critical — Indicates that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority. 35. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 36. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process. 37. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 38. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different

states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

### 39. Defect Life Cycle in Detail

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

#### 39.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

#### 39.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.
- q) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- r) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.
- s) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.
- t) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".
- #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".
- #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".
- #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.
- #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.
- #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.
- #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 39.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 40. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 41. Defect

**Management 41.1. Defect Tracking** In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

**41.1.1. Objectives of Defect Tracking** A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

**41.1.2. Why is Defect Tracking Necessary?** o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.

o Monitors the bugs that had already been resolved, thereby saving time.

o To ensure that right defects are being worked on.

**41.2. Testing stages and defects Responsibilities** SL No. Testing Stage Environment Defect Owner Defect Retested By

Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST

Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team

4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager

**41.3. Defect Triage Calls** Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

**41.3.1. Thumb rules of Triaging** o All reported defects have been reviewed

o All accepted defects have been prioritized

o All accepted defects have severity attached

o All rejected defects should have reasonable explanations for the testing team

o Every defect has been assigned to an appropriate owner, individual or team

o Root cause analysis for every accepted defect has been done

**41.3.2. How does Defect Triaging work?** The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution

- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

**41.4. Defect Metrics** Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs

have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

41.5. Defect Reporting Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

41.5.1. Defect Trends by Release o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

41.5.2. Defects Density o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

41.5.3. Defect Fix Rate per Release o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

41.5.4. Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

41.5.5. Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish. Follow the steps below to create a fishbone diagram:

- (17) Write the problem at the head of the fish.
- (18) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
- (19) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
- (20) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement.

Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

41.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process

of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

41.6.1. Action items, Action Owners, Tracking, Checklist etc.

42. Defect Data • Name of the Person • Types of Testing • Problem Summary • Detailed Description of Defect. • Steps to Reproduce • Life Cycle Phase • Work product where Defect was introduced. • Severity and Priority • Subsystem or Component where the Defect is introduced. • Project Activity occurs when the Defect is introduced. • Identification Method • Type of Defect • Projects and Products in which problems exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References

43. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first?

Severity Description Examples

- 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached • Customer data loss
- 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted
- 3 A minor incident with low impact • A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

44. Defect Fix timeline

Severity	Response Time	Resolution Time
1	1	2
2	2	3

45. Conclusion

This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

46. What is A Defect?

A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the

responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 47.

**Defect Management Life Cycle** When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project. 48. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again. 48.1. Defect Workflow The diagram below shows the actual life cycle of a defect. 48.2. Defect States #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle. #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer. #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason. u) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer. v) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer. w) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'. x) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug". #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed". #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest". #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8)

Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

48.3. Guidelines for Implementing a Defect Life Cycle Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

49. Goals of Defect Management Process (DMP) Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

50. Defect Management

50.1. Defect Tracking In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

50.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

50.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

50.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Defect Retested By
1	Unit Testing (UT)	DEV	DEV Team	Dev Team
2	System Testing (ST)	ST	ST Team	ST Team
3	System Integration Testing (SIT)	SIT	SIT Team	SIT Team
4	User Acceptance Testing (UAT)	UAT	UAT Team	UAT Team

50.3. Defect Triage Calls

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

50.3.1. Thumb rules of Triaging

- o All reported defects have been

reviewed o All accepted defects have been prioritized o All accepted defects have severity attached o All rejected defects should have reasonable explanations for the testing team o Every defect has been assigned to an appropriate owner, individual or team o Root cause analysis for every accepted defect has been done 50.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

50.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

50.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

50.5.1. Defect Trends by Release

o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

50.5.2. Defects Density

o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

50.5.3. Defect Fix Rate per Release

o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

50.5.4. Defect Aging

o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

50.5.5. Defect Analysis through RCA

o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram

A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the

fish. Follow the steps below to create a fishbone diagram: (21)Write the problem at the head of the fish. (22)Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N] (23)Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N. (24)Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

#### 50.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

##### 50.6.1. Action items, Action Owners, Tracking, Checklist etc.

51. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was resolved and recommendations for testing.
- References

52. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples

- 1 A critical incident with very high impact
- A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss
- 2 A major incident with significant impact
- A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted
- 3 A minor incident with low impact
- A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

Blocker — Highest priority. Indicates that this issue takes precedence over all others.

Critical — Indicates

that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority.

53. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 54. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

55. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

56. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

57. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

57.1. Defect Workflow The diagram below shows the actual life cycle of a defect.

57.2. Defect States #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle. #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA

or test lead of the testing team to a developer. #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason. y) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer. z) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer. aa) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'. bb) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug". #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed". #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest". #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

57.3. Guidelines for Implementing a Defect Life Cycle Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

58. Goals of Defect Management Process (DMP) Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

59. Defect Management

59.1. Defect Tracking In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

59.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to

avoid further defects. o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

### 59.1.2. Why is Defect Tracking Necessary?

o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.

o Monitors the bugs that had already been resolved, thereby saving time.

o To ensure that right defects are being worked on.

### 59.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Defect Retested By
1	Unit Testing (UT)	DEV	DEV Team	Dev Team
2	System Testing (ST)	ST	ST Team	ST Team
3	System Integration Testing (SIT)	SIT	SIT Team	SIT Team
4	User Acceptance Testing (UAT)	UAT	UAT Team	UAT Team

### 59.3. Defect Triage Calls

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

#### 59.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

#### 59.3.2. How does Defect Triaging work?

The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

### 59.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

### 59.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

#### 59.5.1. Defect Trends by Release

- o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and

closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

59.5.2. Defect Density o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

59.5.3. Defect Fix Rate per Release o Defect removal efficiency = (1 - (total defects caught by customer/ total number of defects)) x 100.

o Residual defect density = (total number of defects found by a customer)/ (Total number of defects including customer found defects) x 100.

59.5.4. Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

59.5.5. Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
- Follow the steps below to create a fishbone diagram:

- (25) Write the problem at the head of the fish.
- (26) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
- (27) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
- (28) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement.

Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit.

The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes.

Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

59.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects.

Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

59.6.1. Action items, Action Owners, Tracking, Checklist etc.

60. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect
- Steps to Reproduce
- Life Cycle

Phase • Work product where Defect was introduced. • Severity and Priority • Subsystem or Component where the Defect is introduced. • Project Activity occurs when the Defect is introduced. • Identification Method • Type of Defect • Projects and Products in which problems exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References

61. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples

- 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss
- 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted
- 3 A minor incident with low impact • A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

62. Defect Fix timeline

Severity Response Time Resolution Time 1 2 3

4 63. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

64. What is A Defect?

A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

65. Defect Management Life Cycle

When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the

status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

#### 66. Defect Life Cycle in Detail

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

##### 66.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

##### 66.2. Defect States

#1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.

#2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.

#3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

cc) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.

dd) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.

ee) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.

ff) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".

#4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".

#5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".

#6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.

#7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.

#8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.

#9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

##### 66.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the

status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily. • The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

67. Goals of Defect Management Process (DMP) Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

68. Defect Management

68.1. Defect Tracking In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

68.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

68.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

68.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Defect Retested By
1	Unit Testing (UT)	DEV	DEV Team	Dev Team
2	System Testing (ST)	ST	ST Team	ST Team
3	System Integration Testing (SIT)	SIT	SIT Team	SIT Team
4	User Acceptance Testing (UAT)	UAT	UAT Team	UAT Team
			Test Manager	

68.3. Defect Triage

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

68.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

68.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all

relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

#### 68.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

#### 68.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

##### 68.5.1. Defect Trends by Release

o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

##### 68.5.2. Defects Density

o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

##### 68.5.3. Defect Fix Rate per Release

o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

##### 68.5.4. Defect Aging

o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

##### 68.5.5. Defect Analysis through RCA

o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
- Follow the steps below to create a fishbone diagram:
  - (29) Write the problem at the head of the fish.
  - (30) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
  - (31) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
  - (32) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement.

Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit.

The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem

areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

68.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

68.6.1. Action items, Action Owners, Tracking, Checklist etc.

69. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was resolved and recommendations for testing.
- References

70. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first?

Severity Description Examples

- 1 A critical incident with very high impact
- A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss

- 2 A major incident with significant impact
- A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted

- 3 A minor incident with low impact
- A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

71. Defect Fix timeline

Severity Response Time Resolution Time

Severity	Response Time	Resolution Time
Blocker	1	2
Critical	2	3
Major	3	4
Minor	4	72.

Conclusion

This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The

organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

73. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

74. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

75. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

75.1. Defect Workflow The diagram below shows the actual life cycle of a defect.

75.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

gg)  
Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.

hh) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.

ii) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.

**Defect Management Process** Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

**1. What is A Defect?** A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

**2. Defect Management Life Cycle** When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

**3. Defect Life Cycle in Detail** The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

**3.1. Defect Workflow** The diagram below shows the actual life cycle of a defect.

**3.2. Defect States**

- #1) **New:** This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) **Assigned:** In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) **Open:** Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

a) **Rejected:** If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.

b) **Duplicate:** If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of

the defect is changed to 'Duplicate' by the developer. c) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'. d) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug". #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed". #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest". #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 3.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 4. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 5. Defect Management

#### 5.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

##### 5.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

##### 5.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

#### 5.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Defect Retested By	Defect Tracking By
1	Unit Testing (UT)	DEV	DEV Team	Dev Team	System Testing (ST)
2	System Testing (ST)	ST	ST Team		

ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team 4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager 5.3.Defect Triage Calls Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test. 5.3.1.Thumb rules of Triaging o All reported defects have been reviewed o All accepted defects have been prioritized o All accepted defects have severity attached o All rejected defects should have reasonable explanations for the testing team o Every defect has been assigned to an appropriate owner, individual or team o Root cause analysis for every accepted defect has been done 5.3.2.How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are: • The validity of the defect • Time sensitivity for resolution • The complexity involved in the resolution • Business impact This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced. 5.4.Defect Metrics Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be: • Defect distribution by cause • Defect distribution by feature/functional area • Defect distribution by Severity • Defect distribution by Priority • Defect distribution by type • Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user 5.5.Defect Reporting Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them. 5.5.1.Defect Trends by Release o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed. 5.5.2.Defects Density o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software. o Defect Density = Total Defect/Size of the Release 5.5.3.Defect Fix Rate per Release o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ . o Residual defect density = (total number of defects found by a customer)/ (Total number of defects including customer found

defects) x 100.

5.5.4. Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

5.5.5. Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.

Follow the steps below to create a fishbone diagram:

- (1) Write the problem at the head of the fish.
- (2) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
- (3) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
- (4) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement

Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit.

The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes.

Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

5.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects.

Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

5.6.1. Action items, Action Owners, Tracking, Checklist etc.

6. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was resolved and recommendations for testing.
- References

7. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted 3 A minor incident with low impact • A minor inconvenience to customers, workaround available In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs Blocker — Highest priority. Indicates that this issue takes precedence over all others. Critical — Indicates that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority.

8. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 9.

Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

10. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

11. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle.

Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

### 12. Defect Life Cycle in Detail

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

#### 12.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

#### 12.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.
- e) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- f) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.
- g) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.
- h) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".
- #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".
- #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".
- #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.
- #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.
- #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.
- #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 12.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 13. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 14. Defect Management

#### 14.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by

inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier.

Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

14.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

14.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

14.2. Testing stages and defects Responsibilities SL No. Testing Stage Environment Defect Owner Defect Retested By Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team 4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager

14.3. Defect Triage Calls Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

14.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

14.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

14.4. Defect Metrics Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect

distribution by feature/functional area • Defect distribution by Severity • Defect distribution by Priority • Defect distribution by type • Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

14.5. Defect Reporting Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

14.5.1. Defect Trends by Release o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

14.5.2. Defects Density o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

14.5.3. Defect Fix Rate per Release o Defect removal efficiency = (1 - (total defects caught by customer/ total number of defects)) x 100.

o Residual defect density = (total number of defects found by a customer)/ (Total number of defects including customer found defects) x 100.

14.5.4. Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

14.5.5. Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish. Follow the steps below to create a fishbone diagram:

- (5) Write the problem at the head of the fish.
- (6) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
- (7) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
- (8) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

14.6. Defect Prevention Processes & Improvements Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is

a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

14.6.1. Action items, Action Owners, Tracking, Checklist etc.

15. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was resolved and recommendations for testing.
- References

16. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first?

Severity Description Examples

- 1 A critical incident with very high impact
- A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss

2 A major incident with significant impact

- A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted

3 A minor incident with low impact

- A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

17. Defect Fix timeline

Severity Response Time Resolution Time

1	2	3
---	---	---

4 18. Conclusion

This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

19. What is A Defect?

A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand

the defect life cycle before moving to the workflow and different states of the defect.

**20. Defect Management Life Cycle** When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

**21. Defect Life Cycle in Detail** The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

**21.1. Defect Workflow** The diagram below shows the actual life cycle of a defect.

**21.2. Defect States**

- #1) **New:** This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) **Assigned:** In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) **Open:** Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.
- i) **Rejected:** If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- j) **Duplicate:** If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.
- k) **Deferred:** If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.
- l) **Not a Bug:** If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".
- #4) **Fixed:** When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".
- #5) **Pending Retest:** After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".
- #6) **Retest:** At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.
- #7) **Reopen:** If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.
- #8) **Verified:** If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status

of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 21.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 22. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 23. Defect Management

#### 23.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

##### 23.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

##### 23.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

#### 23.2. Testing stages and defects Responsibilities

SL No. Testing Stage Environment Defect Owner Defect Retested By Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team 4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager

### 23.3. Defect Triage

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

#### 23.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team

Every defect has been assigned to an appropriate owner, individual or team o Root cause analysis for every accepted defect has been done 23.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

### 23.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

### 23.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

#### 23.5.1. Defect Trends by Release

o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

#### 23.5.2. Defects Density

o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

#### 23.5.3. Defect Fix Rate per Release

o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

#### 23.5.4. Defect Aging

o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

#### 23.5.5. Defect Analysis through RCA

o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
- Follow the steps below to create a fishbone diagram:

- (9) Write the problem at the head of the fish.
- (10) Identify the category of causes and write at end of each bone [cause category 1,

cause category 2 ..... cause category N] (11)Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N. (12)Extend the causes to secondary, tertiary, and more levels as applicable. o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

### 23.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

#### 23.6.1. Action items, Action Owners, Tracking, Checklist etc.

24. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was resolved and recommendations for testing.
- References

25. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first?

Severity Description Examples

- 1 A critical incident with very high impact
- A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss

- 2 A major incident with significant impact
- A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted

- 3 A minor incident with low impact
- A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.

Trivial — Lowest priority. 26. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 27. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process. 28. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 29. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project. 30. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again. 30.1. Defect Workflow The diagram below shows the actual life cycle of a defect. 30.2. Defect States #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle. #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer. #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that

the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason. m)

Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer. n) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer. o) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'. p) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug". #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".

#5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest". #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8)

Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

30.3. Guidelines for Implementing a Defect Life Cycle Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

31. Goals of Defect Management Process (DMP) Given below are the various goals of this process:

- Prevent the Defect
- Early Detection

- Minimize the impact
- Resolution of the Defect
- Process improvement

32. Defect Management 32.1. Defect Tracking In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier.

Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

32.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

32.1.2. Why is Defect Tracking

Necessary? o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process. o Monitors the bugs that had already been resolved, thereby saving time. o To ensure that right defects are being worked on.

### 32.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Defect Retested By
1	Unit Testing (UT)	DEV	DEV Team	Dev Team
2	System Testing (ST)	ST	ST	ST
3	System Integration Testing (SIT)	SIT	SIT	Team SIT
4	User Acceptance Testing (UAT)	UAT	UAT Team	UAT Team
				Test Manager

### 32.3. Defect Triage Calls

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the “right” defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

#### 32.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

#### 32.3.2. How does Defect Triaging work?

The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

### 32.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

### 32.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

#### 32.5.1. Defect Trends by Release

- o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

#### 32.5.2. Defects Density

- o Defect density can be defined as

the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

- o Defect Density = Total Defect/Size of the Release
- 32.5.3. Defect Fix Rate per Release
- o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .
- o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

32.5.4. Defect Aging

- o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

32.5.5. Defect Analysis through RCA

- o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.
- o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
- Follow the steps below to create a fishbone diagram:
- (13) Write the problem at the head of the fish.
- (14) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
- (15) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
- (16) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement

Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit.

The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes.

Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

32.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects.

Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

32.6.1. Action items, Action Owners, Tracking, Checklist etc.

33. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is

introduced. • Identification Method • Type of Defect • Projects and Products in which problems exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References

34. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples

1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss

2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted

3 A minor incident with low impact • A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

Blocker — Highest priority. Indicates that this issue takes precedence over all others.

Critical — Indicates that this issue is causing a problem and requires urgent attention.

Major — Indicates that this issue has a significant impact.

Minor — Indicates that this issue has a relatively minor impact.

Trivial — Lowest priority.

35. Defect Fix timeline

Severity Response Time Resolution Time 1 2 3 4

36. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

37. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

38. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected,

postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

### 39. Defect Life Cycle in Detail

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

#### 39.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

#### 39.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.
- q) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- r) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.
- s) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.
- t) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".
- #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".
- #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".
- #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.
- #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.
- #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.
- #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

#### 39.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The

defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

#### 40. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

#### 41. Defect Management

##### 41.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

##### 41.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

##### 41.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

#### 41.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Defect Retested By
1	Unit Testing (UT)	DEV	DEV Team	Dev Team
2	System Testing (ST)	ST	ST Team	ST Team
3	System Integration Testing (SIT)	SIT	SIT Team	SIT Team
4	User Acceptance Testing (UAT)	UAT	UAT Team	UAT Team

#### 41.3. Defect Triage Calls

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

##### 41.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

##### 41.3.2. How does Defect Triaging work?

The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every

single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

#### 41.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

#### 41.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

##### 41.5.1. Defect Trends by Release

- o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

##### 41.5.2. Defects Density

- o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.
- o Defect Density = Total Defect/Size of the Release

##### 41.5.3. Defect Fix Rate per Release

- o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .
- o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

##### 41.5.4. Defect Aging

- o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

##### 41.5.5. Defect Analysis through RCA

- o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.
- o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.
- Steps to create a Fishbone Diagram:
  - Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
  - Follow the steps below to create a fishbone diagram:
    - (17) Write the problem at the head of the fish.
    - (18) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
    - (19) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
    - (20) Extend the causes to secondary, tertiary, and more levels as applicable.
- o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement. Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads

more effectively. • Improving productivity. • Improving profitability. Pareto Analysis Steps 1. Identify and List Problems 2. Identify the Root Cause of Each Problem 3. Score Problems 4. Group Problems Together 5. Add up Scores for Each Group 6. Take Action 41.6. Defect Prevention Processes & Improvements Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks: #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products. #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system. #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects. #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis 41.6.1. Action items, Action Owners, Tracking, Checklist etc. 42. Defect Data • Name of the Person • Types of Testing • Problem Summary • Detailed Description of Defect. • Steps to Reproduce • Life Cycle Phase • Work product where Defect was introduced. • Severity and Priority • Subsystem or Component where the Defect is introduced. • Project Activity occurs when the Defect is introduced. • Identification Method • Type of Defect • Projects and Products in which problems exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References 43. Defect Priority and Severity Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted 3 A minor incident with low impact • A minor inconvenience to customers, workaround available In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs Blocker — Highest priority. Indicates that this issue takes precedence over all others. Critical — Indicates that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority. 44. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 45. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process. 46. What is A Defect? A Defect, in

simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 47.

**Defect Management Life Cycle** When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project. 48.

**Defect Life Cycle in Detail** The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again. 48.1. Defect Workflow

The diagram below shows the actual life cycle of a defect. 48.2. Defect States #1) New:

This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle. #2) Assigned:

In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer. #3) Open:

Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below

namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason. u) Rejected:

If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer. v) Duplicate:

If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer. w) Deferred:

If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.

x) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug". #4) Fixed:

When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".

#5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of

the defect remains in "Pending Retest". #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 48.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 49. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 50. Defect Management

#### 50.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

##### 50.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

##### 50.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

#### 50.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Retested By
1	Unit Testing (UT)	DEV	DEV Team	Dev Team
2	System Testing (ST)	ST	ST Team	ST Team
3	System Integration Testing (SIT)	SIT	SIT Team	SIT Team
4	User Acceptance Testing (UAT)	UAT	UAT Team	UAT Team

#### 50.3. Defect Triage

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in

preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

50.3.1. Thumb rules of Triaging o All reported defects have been reviewed o All accepted defects have been prioritized o All accepted defects have severity attached o All rejected defects should have reasonable explanations for the testing team o Every defect has been assigned to an appropriate owner, individual or team o Root cause analysis for every accepted defect has been done

50.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

50.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

50.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

50.5.1. Defect Trends by Release

o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

50.5.2. Defects Density

o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

50.5.3. Defect Fix Rate per Release

o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

50.5.4. Defect Aging

o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

50.5.5. Defect Analysis through RCA

o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in

sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms. Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish. Follow the steps below to create a fishbone diagram: (21)Write the problem at the head of the fish. (22)Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N] (23)Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N. (24)Extend the causes to secondary, tertiary, and more levels as applicable. o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

#### 50.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

##### 50.6.1. Action items, Action Owners, Tracking, Checklist etc.

51. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced
- Severity and Priority
- Subsystem or Component where the Defect is introduced
- Project Activity occurs when the Defect is introduced
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect
- Dates when various defect lifecycle phases occur
- Description of how the defect was resolved and recommendations for testing
- References

52. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples

- 1 A critical incident with very high impact
- A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss
- 2 A major incident with significant impact
- A customer-facing

service is unavailable for a subset of customers • Core functionality is significantly impacted 3 A minor incident with low impact • A minor inconvenience to customers, workaround available In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs Blocker — Highest priority. Indicates that this issue takes precedence over all others. Critical — Indicates that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority. 53. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 54. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process. 55. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 56. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project. 57. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again. 57.1. Defect Workflow The diagram below shows the actual life cycle of a defect. 57.2.

Defect States #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle. #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer. #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason. y) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer. z) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer. aa) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'. bb) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug". #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed". #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest". #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

57.3. Guidelines for Implementing a Defect Life Cycle Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

58. Goals of Defect Management Process (DMP) Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

59. Defect Management

59.1. Defect Tracking In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

59.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

59.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

59.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Defect Retested By
1	Unit Testing (UT)	DEV	DEV Team	Dev Team
2	System Testing (ST)	ST	ST Team	Team ST
3	System Integration Testing (SIT)	SIT	SIT Team	SIT Team
4	User Acceptance Testing (UAT)	UAT	UAT Team	UAT Team

59.3. Defect Triage Calls

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

59.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

59.3.2. How does Defect Triaging work?

The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

59.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

59.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management

board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

59.5.1. Defect Trends by Release o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

59.5.2. Defects Density o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

59.5.3. Defect Fix Rate per Release o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

59.5.4. Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

59.5.5. Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
- Follow the steps below to create a fishbone diagram:
  - (25) Write the problem at the head of the fish.
  - (26) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
  - (27) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
  - (28) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit.

The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

59.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can

be used to support analyzing defects. #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

59.6.1. Action items, Action Owners, Tracking, Checklist etc.

60. Defect Data • Name of the Person • Types of Testing • Problem Summary • Detailed Description of Defect. • Steps to Reproduce • Life Cycle Phase • Work product where Defect was introduced. • Severity and Priority • Subsystem or Component where the Defect is introduced. • Project Activity occurs when the Defect is introduced. • Identification Method • Type of Defect • Projects and Products in which problems exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References

61. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples

1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss

2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted

3 A minor incident with low impact • A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

Blocker — Highest priority. Indicates that this issue takes precedence over all others. Critical — Indicates that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority.

62. Defect Fix timeline

Severity Response Time Resolution Time 1 2 3

4 63. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process

Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

64. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

65. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a

flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

### 66. Defect Life Cycle in Detail

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

#### 66.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

#### 66.2. Defect States #1) New:

This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.

#### #2) Assigned:

In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.

#### #3) Open:

Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

cc)

#### Rejected:

If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.

#### dd) Duplicate:

If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.

#### ee) Deferred:

If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.

#### ff) Not a Bug:

If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".

#### #4) Fixed:

When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".

#### #5) Pending Retest:

After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".

#### #6) Retest:

At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.

#### #7) Reopen:

If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.

#### #8) Verified:

If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.

#### #9) Closed:

When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 66.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be

properly documented to avoid any confusion in the future. • Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results. • Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily. • The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

67. Goals of Defect Management Process (DMP) Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

68. Defect Management

68.1. Defect Tracking In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

68.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

68.1.2. Why is Defect Tracking Necessary?

- A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- Monitors the bugs that had already been resolved, thereby saving time.
- To ensure that right defects are being worked on.

68.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Defect Retested By
1	Unit Testing (UT)	DEV	DEV Team	Dev Team
2	System Testing (ST)	ST	ST Team	ST Team
3	System Integration Testing (SIT)	SIT	SIT Team	SIT Team
4	User Acceptance Testing (UAT)	UAT	UAT Team	UAT Team

68.3. Defect Triage

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the “right” defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

68.3.1. Thumb rules of Triaging

- All reported defects have been reviewed
- All accepted defects have been prioritized
- All accepted defects have severity attached
- All rejected defects should have reasonable explanations for the testing team
- Every defect has been assigned to an appropriate owner, individual or team
- Root cause analysis for every accepted defect has been done

68.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected

at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

#### 68.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

#### 68.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

##### 68.5.1. Defect Trends by Release

o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

##### 68.5.2. Defects Density

o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

##### 68.5.3. Defect Fix Rate per Release

o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

##### 68.5.4. Defect Aging

o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

##### 68.5.5. Defect Analysis through RCA

o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
- Follow the steps below to create a fishbone diagram:
  - (29) Write the problem at the head of the fish.
  - (30) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
  - (31) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
  - (32) Extend the causes to secondary, tertiary, and more levels as applicable.
- o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement.

Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to

focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

68.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

68.6.1. Action items, Action Owners, Tracking, Checklist etc.

69. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was resolved and recommendations for testing.
- References

70. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first?

Severity Description Examples

- 1 A critical incident with very high impact
- A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss
- 2 A major incident with significant impact
- A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted
- 3 A minor incident with low impact
- A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

71. Defect Fix timeline

Severity Response Time Resolution Time

1	2	3
4	72. Conclusion	This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process

Defect management process is mainly used to improve the

efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

73. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

74. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

75. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

75.1. Defect Workflow The diagram below shows the actual life cycle of a defect.

75.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

gg)  
Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.  
hh) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.  
ii) Deferred: If the developer feels that

the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.

**Defect Management Process** Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

1. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

2. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

3. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

3.1. Defect Workflow The diagram below shows the actual life cycle of a defect.

3.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the

developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

a) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.

b) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.

c) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.

d) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".

#4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".

#5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".

#6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.

#7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.

#8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.

#9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 3.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 4. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 5. Defect Management

#### 5.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

##### 5.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the

work. Also, it enhances the efficient work delivered. 5.1.2.Why is Defect Tracking Necessary? o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process. o Monitors the bugs that had already been resolved, thereby saving time. o To ensure that right defects are being worked on. 5.2.Testing stages and defects

Responsibilities SL No. Testing Stage Environment Defect Owner Defect Retested By Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team 4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager 5.3.Defect Triage Calls Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test. 5.3.1.Thumb rules of Triaging o All reported defects have been reviewed o All accepted defects have been prioritized o All accepted defects have severity attached o All rejected defects should have reasonable explanations for the testing team o Every defect has been assigned to an appropriate owner, individual or team o Root cause analysis for every accepted defect has been done 5.3.2.How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are: • The validity of the defect • Time sensitivity for resolution • The complexity involved in the resolution • Business impact This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced. 5.4.Defect Metrics Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be: • Defect distribution by cause • Defect distribution by feature/functional area • Defect distribution by Severity • Defect distribution by Priority • Defect distribution by type • Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user 5.5.Defect Reporting Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them. 5.5.1.Defect Trends by Release o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which

the defects are processed.

5.5.2. Defect Density o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

5.5.3. Defect Fix Rate per Release o Defect removal efficiency = (1 - (total defects caught by customer/ total number of defects)) x 100.

o Residual defect density = (total number of defects found by a customer)/ (Total number of defects including customer found defects) x 100.

5.5.4. Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

5.5.5. Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.

Follow the steps below to create a fishbone diagram:

- (1) Write the problem at the head of the fish.
- (2) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
- (3) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
- (4) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement

Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit.

The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes.

Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

5.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects.

Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

5.6.1. Action items, Action Owners, Tracking, Checklist etc.

6. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component

where the Defect is introduced. • Project Activity occurs when the Defect is introduced. • Identification Method • Type of Defect • Projects and Products in which problems exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.

• Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References

### 7. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples

- 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached • Customer data loss
- 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted
- 3 A minor incident with low impact • A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

### 8. Defect Fix timeline Severity Response Time Resolution Time

1	2	3
4	9.	Conclusion

This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process

Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

### 10. What is A Defect?

A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

### 11. Defect Management Life Cycle

When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as

expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

### 12. Defect Life Cycle in Detail

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

#### 12.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

#### 12.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.
- e) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- f) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.
- g) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.
- h) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".
- #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".
- #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".
- #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.
- #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.
- #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.
- #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 12.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The

defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 13. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 14. Defect Management

#### 14.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

#### 14.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

#### 14.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

### 14.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Defect Retested By
1	Unit Testing (UT)	DEV	DEV Team	Dev Team
2	System Testing (ST)	ST	ST Team	ST Team
3	System Integration Testing (SIT)	SIT	SIT Team	SIT Team
4	User Acceptance Testing (UAT)	UAT	UAT Team	UAT Team

### 14.3. Defect Triage Calls

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

#### 14.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

#### 14.3.2. How does Defect Triaging work?

The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every

single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

#### 14.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

#### 14.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

##### 14.5.1. Defect Trends by Release

- o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

##### 14.5.2. Defects Density

- o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.
- o Defect Density = Total Defect/Size of the Release

##### 14.5.3. Defect Fix Rate per Release

- o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .
- o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

##### 14.5.4. Defect Aging

- o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

##### 14.5.5. Defect Analysis through RCA

- o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.
- o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.
- Steps to create a Fishbone Diagram:
  - Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
  - Follow the steps below to create a fishbone diagram:
    - (5) Write the problem at the head of the fish.
    - (6) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
    - (7) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
    - (8) Extend the causes to secondary, tertiary, and more levels as applicable.
- o Pareto charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement. Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads

more effectively. • Improving productivity. • Improving profitability. Pareto Analysis Steps 1. Identify and List Problems 2. Identify the Root Cause of Each Problem 3. Score Problems 4. Group Problems Together 5. Add up Scores for Each Group 6. Take Action 14.6. Defect Prevention Processes & Improvements Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks: #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products. #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system. #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects. #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis 14.6.1. Action items, Action Owners, Tracking, Checklist etc. 15. Defect Data • Name of the Person • Types of Testing • Problem Summary • Detailed Description of Defect. • Steps to Reproduce • Life Cycle Phase • Work product where Defect was introduced. • Severity and Priority • Subsystem or Component where the Defect is introduced. • Project Activity occurs when the Defect is introduced. • Identification Method • Type of Defect • Projects and Products in which problems exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References 16. Defect Priority and Severity Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted 3 A minor incident with low impact • A minor inconvenience to customers, workaround available In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs Blocker — Highest priority. Indicates that this issue takes precedence over all others. Critical — Indicates that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority. 17. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 18. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process. 19. What is A Defect? A Defect, in

simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 20.

**Defect Management Life Cycle** When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project. 21. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again. 21.1. Defect Workflow The diagram below shows the actual life cycle of a defect. 21.2. Defect States #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle. #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer. #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason. i) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer. j) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer. k) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'. l) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug". #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed". #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of

the defect remains in "Pending Retest". #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 21.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 22. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 23. Defect Management

#### 23.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

##### 23.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

##### 23.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

#### 23.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Retested By
1	Unit Testing (UT)	DEV	DEV Team	Dev Team
2	System Testing (ST)	ST	ST Team	ST Team
3	System Integration Testing (SIT)	SIT	SIT Team	SIT Team
4	User Acceptance Testing (UAT)	UAT	UAT Team	UAT Team

#### 23.3. Defect Triage

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in

preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

23.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

23.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

23.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

23.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

23.5.1. Defect Trends by Release

- o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

23.5.2. Defects Density

- o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.
- o Defect Density = Total Defect/Size of the Release

23.5.3. Defect Fix Rate per Release

- o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .
- o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

23.5.4. Defect Aging

- o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

23.5.5. Defect Analysis through RCA

- o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.
- o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in

sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms. Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish. Follow the steps below to create a fishbone diagram: (9) Write the problem at the head of the fish. (10) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N] (11) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N. (12) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement

Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

### 23.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

#### 23.6.1. Action items, Action Owners, Tracking, Checklist etc.

24. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced
- Severity and Priority
- Subsystem or Component where the Defect is introduced
- Project Activity occurs when the Defect is introduced
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect
- Dates when various defect lifecycle phases occur
- Description of how the defect was resolved and recommendations for testing
- References

#### 25. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first?

Severity Description Examples

- 1 A critical incident with very high impact
- A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss
- 2 A major incident with significant impact
- A customer-facing

service is unavailable for a subset of customers • Core functionality is significantly impacted 3 A minor incident with low impact • A minor inconvenience to customers, workaround available In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs Blocker — Highest priority. Indicates that this issue takes precedence over all others. Critical — Indicates that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority. 26. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 27. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process. 28. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 29. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project. 30. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again. 30.1. Defect Workflow The diagram below shows the actual life cycle of a defect. 30.2.

Defect States #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle. #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer. #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason. m) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer. n) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer. o) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'. p) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug". #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed". #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest". #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 30.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 31. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 32. Defect Management

#### 32.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

32.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

32.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

32.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Defect Retested By
1	Unit Testing (UT)	DEV	DEV Team	Dev Team
2	System Testing (ST)	ST	ST Team	Team ST
3	System Integration Testing (SIT)	SIT	SIT Team	SIT Team
4	User Acceptance Testing (UAT)	UAT	UAT Team	UAT Team

32.3. Defect Triage Calls

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

32.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

32.3.2. How does Defect Triaging work?

The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

32.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

32.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management

board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

32.5.1. Defect Trends by Release o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

32.5.2. Defects Density o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

32.5.3. Defect Fix Rate per Release o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

32.5.4. Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

32.5.5. Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
- Follow the steps below to create a fishbone diagram:
  - (13) Write the problem at the head of the fish.
  - (14) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
  - (15) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
  - (16) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit.

The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

32.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can

be used to support analyzing defects. #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

32.6.1. Action items, Action Owners, Tracking, Checklist etc.

33. Defect Data • Name of the Person • Types of Testing • Problem Summary • Detailed Description of Defect. • Steps to Reproduce • Life Cycle Phase • Work product where Defect was introduced. • Severity and Priority • Subsystem or Component where the Defect is introduced. • Project Activity occurs when the Defect is introduced. • Identification Method • Type of Defect • Projects and Products in which problems exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References

34. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples

1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss

2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted

3 A minor incident with low impact • A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

Blocker — Highest priority. Indicates that this issue takes precedence over all others. Critical — Indicates that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority.

35. Defect Fix timeline

Severity	Response Time	Resolution Time
1	1	2
2	2	3
3	3	4

36. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process

Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

37. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

38. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a

flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

### 39. Defect Life Cycle in Detail

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

#### 39.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

#### 39.2. Defect States

#1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.

#2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.

#3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

q) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.

r) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.

s) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.

t) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".

#4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".

#5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".

#6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.

#7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.

#8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.

#9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

#### 39.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be

properly documented to avoid any confusion in the future. • Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results. • Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily. • The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

#### 40. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

#### 41. Defect Management

##### 41.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

##### 41.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

##### 41.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

##### 41.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Defect Retested By
1	Unit Testing (UT)	DEV	DEV Team	Dev Team
2	System Testing (ST)	ST	ST Team	ST Team
3	System Integration Testing (SIT)	SIT	SIT Team	SIT Team
4	User Acceptance Testing (UAT)	UAT	UAT Team	UAT Team

##### 41.3. Defect Triage

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the “right” defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

##### 41.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

##### 41.3.2. How does Defect Triaging work?

The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected

at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

#### 41.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

#### 41.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

##### 41.5.1. Defect Trends by Release

o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

##### 41.5.2. Defects Density

o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

##### 41.5.3. Defect Fix Rate per Release

o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

##### 41.5.4. Defect Aging

o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

##### 41.5.5. Defect Analysis through RCA

o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
- Follow the steps below to create a fishbone diagram:
  - (17) Write the problem at the head of the fish.
  - (18) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
  - (19) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
  - (20) Extend the causes to secondary, tertiary, and more levels as applicable.
- o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement.

Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to

focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

#### 41.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

#### 41.6.1. Action items, Action Owners, Tracking, Checklist etc.

42. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was resolved and recommendations for testing.
- References

#### 43. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first?

Severity Description Examples

- 1 A critical incident with very high impact
- A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss
- 2 A major incident with significant impact
- A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted
- 3 A minor incident with low impact
- A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

#### 44. Defect Fix timeline

Severity Response Time Resolution Time

1	2	3
---	---	---

#### 45. Conclusion

This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process

Defect management process is mainly used to improve the

efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

46. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

47. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

48. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

48.1. Defect Workflow The diagram below shows the actual life cycle of a defect.

48.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.
- u) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- v) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.
- w) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a

case, developer can change the status of the defect as 'Deferred'. x) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug". #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed". #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest". #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 48.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 49. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 50. Defect Management

#### 50.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

##### 50.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

##### 50.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

#### 50.2. Testing stages and defects Responsibilities

SL No. Testing Stage Environment Defect Owner Defect Retested By

Defect Tracking	By 1 Unit Testing (UT)	DEV DEV Team	Dev Team	2 System Testing (ST)	ST	ST Team
ST Team	ST Team	3 System Integration Testing (SIT)	SIT	SIT Team	SIT Team	SIT Team
4 User Acceptance Testing (UAT)	UAT	UAT Team	UAT Team	Test Manager		

### 50.3. Defect Triage

Calls Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

50.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

50.3.2. How does Defect Triaging work?

The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

50.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

50.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

50.5.1. Defect Trends by Release

- o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

50.5.2. Defects Density

- o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.
- o 
$$\text{Defect Density} = \frac{\text{Total Defect}}{\text{Size of the Release}}$$

50.5.3. Defect Fix Rate per Release

- o 
$$\text{Defect removal efficiency} = (1 - \frac{\text{total defects caught by customer}}{\text{total number of defects}}) \times 100$$

o Residual defect density = 
$$\frac{\text{total number of defects found by a customer}}{\text{Total number of defects including customer found defects}} \times 100$$

50.5.4. Defect Aging

- o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the

defect was fixed (if the defect is already fixed). 50.5.5. Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization. o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms. Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish. Follow the steps below to create a fishbone diagram: (21)Write the problem at the head of the fish. (22)Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N] (23)Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N. (24)Extend the causes to secondary, tertiary, and more levels as applicable. o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

50.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

50.6.1. Action items, Action Owners, Tracking, Checklist etc.

51. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was resolved and recommendations for testing.
- References

52. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does

an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted 3 A minor incident with low impact • A minor inconvenience to customers, workaround available In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs Blocker — Highest priority. Indicates that this issue takes precedence over all others. Critical — Indicates that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority.

53. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 54. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

55. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

56. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is

closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

### 57. Defect Life Cycle in Detail

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

#### 57.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

#### 57.2. Defect States

#1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.

#2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.

#3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

y) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.

z) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.

aa) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.

bb) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".

#4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".

#5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".

#6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.

#7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.

#8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.

#9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 57.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 58. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 59. Defect Management

#### 59.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the

product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier.

Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

59.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

59.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

59.2. Testing stages and defects Responsibilities SL No. Testing Stage Environment Defect Owner Defect Retested By Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team 4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager

59.3. Defect Triage Calls Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

59.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

59.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level.

Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

59.4. Defect Metrics Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by

Priority • Defect distribution by type • Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

59.5. Defect Reporting Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

59.5.1. Defect Trends by Release o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

59.5.2. Defects Density o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

59.5.3. Defect Fix Rate per Release o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

59.5.4. Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

59.5.5. Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish. Follow the steps below to create a fishbone diagram:

- (25) Write the problem at the head of the fish.
- (26) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
- (27) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
- (28) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement

Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit.

The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes.

Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

59.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the

diagram below is pretty much half of our testing tasks: #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products. #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system. #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects. #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

59.6.1. Action items, Action Owners, Tracking, Checklist etc.

60. Defect Data • Name of the Person • Types of Testing • Problem Summary • Detailed Description of Defect. • Steps to Reproduce • Life Cycle Phase • Work product where Defect was introduced. • Severity and Priority • Subsystem or Component where the Defect is introduced. • Project Activity occurs when the Defect is introduced. • Identification Method • Type of Defect • Projects and Products in which problems exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References

61. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples

1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss

2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted

3 A minor incident with low impact • A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

Blocker — Highest priority. Indicates that this issue takes precedence over all others.

Critical — Indicates that this issue is causing a problem and requires urgent attention.

Major — Indicates that this issue has a significant impact.

Minor — Indicates that this issue has a relatively minor impact.

Trivial — Lowest priority.

62. Defect Fix timeline

Severity Response Time Resolution Time 1 2 3 4

63. Conclusion

This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process

Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

64. What is A Defect?

A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

65.

**Defect Management Life Cycle** When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

**66. Defect Life Cycle in Detail** The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

**66.1. Defect Workflow** The diagram below shows the actual life cycle of a defect.

**66.2. Defect States**

- #1) **New:** This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) **Assigned:** In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) **Open:** Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.
- cc)  
**Rejected:** If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- dd)  
**Duplicate:** If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.
- ee)  
**Deferred:** If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.
- ff)  
**Not a Bug:** If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".
- #4) **Fixed:** When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".
- #5) **Pending Retest:** After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".
- #6) **Retest:** At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.
- #7) **Reopen:** If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.
- #8) **Verified:** If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.
- #9) **Closed:** When the defect does not exist any

longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 66.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 67. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 68. Defect Management

#### 68.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

##### 68.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

##### 68.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

#### 68.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Defect Retested By
1	Unit Testing (UT)	DEV Team	Dev Team	System Testing (ST) ST
2	System Testing (ST)	Team ST	Team ST	Team SIT
3	System Integration Testing (SIT)	SIT	SIT Team	SIT Team
4	User Acceptance Testing (UAT)	UAT Team	UAT Team	Test Manager

#### 68.3. Defect Triage

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

##### 68.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause

analysis for every accepted defect has been done 68.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

#### 68.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

#### 68.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

##### 68.5.1. Defect Trends by Release

o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

##### 68.5.2. Defects Density

o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

##### 68.5.3. Defect Fix Rate per Release

o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

##### 68.5.4. Defect Aging

o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

##### 68.5.5. Defect Analysis through RCA

o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
- Follow the steps below to create a fishbone diagram:
  - (29) Write the problem at the head of the fish.
  - (30) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
  - (31) Identify the primary causes under each

category and mark it as primary cause 1, primary cause 2, primary cause N. (32) Extend the causes to secondary, tertiary, and more levels as applicable. o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

#### 68.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

#### 68.6.1. Action items, Action Owners, Tracking, Checklist etc.

69. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was resolved and recommendations for testing.
- References

#### 70. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples

- 1 A critical incident with very high impact
- A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss
- 2 A major incident with significant impact
- A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted
- 3 A minor incident with low impact
- A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

#### 71. Defect Fix timeline Severity Response Time Resolution Time

1 2 3

4 72. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

73. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

74. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

75. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

75.1. Defect Workflow The diagram below shows the actual life cycle of a defect.

75.2. Defect States #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.

#2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.

#3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below

namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason. gg)

Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer. hh) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer. ii) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.

Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

1. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

2. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle.

Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

3. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

3.1.Defect Workflow The diagram below shows the actual life cycle of a defect.

3.2.Defect States #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.

#2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.

#3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the

developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

- a) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- b) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.
- c) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.
- d) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".

- #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".
- #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".
- #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.
- #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.
- #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.
- #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 3.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 4. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 5. Defect Management

#### 5.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

##### 5.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

##### 5.1.2. Why is Defect Tracking Necessary?

- o

A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.

- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

### 5.2. Testing stages and defects

Responsibilities SL No. Testing Stage Environment Defect Owner Defect Retested By Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team 4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager

### 5.3. Defect Triage Calls

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

#### 5.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

#### 5.3.2. How does Defect Triaging work?

The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

### 5.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

### 5.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

#### 5.5.1. Defect Trends by Release

- o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

#### 5.5.2. Defects Density

- o Defect density can be defined as the

number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

- o Defect Density = Total Defect/Size of the Release
- 5.5.3. Defect Fix Rate per Release
- o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .
- o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

5.5.4. Defect Aging

- o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

5.5.5. Defect Analysis through RCA

- o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.
- o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.

Follow the steps below to create a fishbone diagram:

- (1) Write the problem at the head of the fish.
- (2) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
- (3) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
- (4) Extend the causes to secondary, tertiary, and more levels as applicable.

o Pareto charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement

Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit.

The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes.

Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

5.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects.

Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

5.6.1. Action items, Action Owners, Tracking, Checklist etc.

6. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.

Identification Method • Type of Defect • Projects and Products in which problems exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.

- Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References

7. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted 3 A minor incident with low impact • A minor inconvenience to customers, workaround available In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs Blocker — Highest priority. Indicates that this issue takes precedence over all others. Critical — Indicates that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority.

8. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 9. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

10. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

11. Defect Management Life Cycle

When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the

above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

### 12. Defect Life Cycle in Detail

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

#### 12.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

#### 12.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.
- e) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- f) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.
- g) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.
- h) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".
- #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".
- #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".
- #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.
- #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.
- #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.
- #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 12.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects

and thus, in the workflow of the Defect Life Cycle.

### 13. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 14. Defect Management

#### 14.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

#### 14.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

#### 14.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

#### 14.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Defect Retested By
1	Unit Testing (UT)	DEV	DEV Team	Dev Team
2	System Testing (ST)	ST	ST Team	ST Team
3	System Integration Testing (SIT)	SIT	SIT Team	SIT Team
4	User Acceptance Testing (UAT)	UAT	UAT Team	UAT Team
			Test Manager	

#### 14.3. Defect Triage

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

#### 14.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

#### 14.3.2. How does Defect Triaging work?

The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which

ensures that chances of getting a similar defect are significantly reduced.

#### 14.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

#### 14.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

##### 14.5.1. Defect Trends by Release

- The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

##### 14.5.2. Defects Density

- Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.
- Defect Density = Total Defect/Size of the Release

##### 14.5.3. Defect Fix Rate per Release

- Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .
- Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

##### 14.5.4. Defect Aging

- Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

##### 14.5.5. Defect Analysis through RCA

- RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.
- Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.
- Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish. Follow the steps below to create a fishbone diagram:
  - (5) Write the problem at the head of the fish.
  - (6) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
  - (7) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
  - (8) Extend the causes to secondary, tertiary, and more levels as applicable.
- Pareto charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:
- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Identify and List Problems 2. Identify the Root Cause of Each Problem 3. Score Problems 4. Group Problems Together 5. Add up Scores for Each Group 6. Take Action 14.6. Defect Prevention Processes & Improvements Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks: #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products. #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system. #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects. #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis 14.6.1. Action items, Action Owners, Tracking, Checklist etc. 15. Defect Data • Name of the Person • Types of Testing • Problem Summary • Detailed Description of Defect. • Steps to Reproduce • Life Cycle Phase • Work product where Defect was introduced. • Severity and Priority • Subsystem or Component where the Defect is introduced. • Project Activity occurs when the Defect is introduced. • Identification Method • Type of Defect • Projects and Products in which problems exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References 16. Defect Priority and Severity Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted 3 A minor incident with low impact • A minor inconvenience to customers, workaround available In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs Blocker — Highest priority. Indicates that this issue takes precedence over all others. Critical — Indicates that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority. 17. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 18. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process. 19. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an

application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 20.

**Defect Management Life Cycle** When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project. 21.

**Defect Life Cycle in Detail** The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

**21.1. Defect Workflow** The diagram below shows the actual life cycle of a defect.

**21.2. Defect States #1) New:** This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle. **#2) Assigned:** In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer. **#3) Open:** Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason. **i) Rejected:** If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer. **j) Duplicate:** If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer. **k) Deferred:** If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'. **l) Not a Bug:** If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug". **#4) Fixed:** When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed". **#5) Pending Retest:** After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest". **#6) Retest:** At this point, the tester starts the task of

retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 21.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 22. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 23. Defect Management

#### 23.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

##### 23.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

##### 23.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

### 23.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Retested By
1	Unit Testing (UT)	DEV	DEV Team	Dev Team
2	System Testing (ST)	ST Team	ST Team	ST Team
3	System Integration Testing (SIT)	SIT Team	SIT Team	SIT Team
4	User Acceptance Testing (UAT)	UAT Team	UAT Team	Test Manager

### 23.3. Defect Triage Calls

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing

them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

23.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

23.3.2. How does Defect Triaging work?

The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

23.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

23.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

23.5.1. Defect Trends by Release

- o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

23.5.2. Defects Density

- o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.
- o Defect Density = Total Defect/Size of the Release

23.5.3. Defect Fix Rate per Release

- o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .
- o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

23.5.4. Defect Aging

- o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

23.5.5. Defect Analysis through RCA

- o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.
- o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and

effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms. Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish. Follow the steps below to create a fishbone diagram: (9) Write the problem at the head of the fish. (10) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N] (11) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N. (12) Extend the causes to secondary, tertiary, and more levels as applicable.

o Pareto charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

### 23.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

#### 23.6.1. Action items, Action Owners, Tracking, Checklist etc.

24. Defect Data
  - Name of the Person
  - Types of Testing
  - Problem Summary
  - Detailed Description of Defect.
  - Steps to Reproduce
  - Life Cycle Phase
  - Work product where Defect was introduced.
  - Severity and Priority
  - Subsystem or Component where the Defect is introduced.
  - Project Activity occurs when the Defect is introduced.
  - Identification Method
  - Type of Defect
  - Projects and Products in which problems exist
  - Current Owner
  - Current State of the Report
  - Work product where Defect occurred.
  - Impact on Project
  - Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
  - Dates when various defect lifecycle phases occur.
  - Description of how the defect was resolved and recommendations for testing.
  - References
25. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples

- 1 A critical incident with very high impact
  - A customer-facing service, like Jira Cloud, is down for all customers
  - Confidentiality or privacy is breached
  - Customer data loss
  - A major incident with significant impact
  - A customer-facing service is unavailable for a subset of customers
  - Core functionality is significantly impacted
- 3

A minor incident with low impact • A minor inconvenience to customers, workaround available In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs Blocker — Highest priority. Indicates that this issue takes precedence over all others. Critical — Indicates that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority.

26. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 27. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

28. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

29. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

30. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

30.1. Defect Workflow The diagram below shows the actual life cycle of a defect.

30.2. Defect States #1) New: This is the first state of a defect in the Defect Life Cycle. When any

new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle. #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer. #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason. m) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer. n) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer. o) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'. p) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug". #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed". #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest". #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 30.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 31. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 32. Defect Management

#### 32.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

#### 32.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry

as it provides a faster method of correcting the defects o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction. o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects. o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

### 32.1.2. Why is Defect Tracking Necessary?

o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process. o Monitors the bugs that had already been resolved, thereby saving time. o To ensure that right defects are being worked on.

### 32.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Defect Retested By
1	Unit Testing (UT)	DEV	DEV Team	Dev Team
2	System Testing (ST)	ST	ST Team	ST Team
3	System Integration Testing (SIT)	SIT	SIT Team	SIT Team
4	User Acceptance Testing (UAT)	UAT	UAT Team	UAT Team
			Test Manager	

### 32.3. Defect Triage

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

#### 32.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

#### 32.3.2. How does Defect Triaging work?

The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

### 32.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

### 32.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management

process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

32.5.1. Defect Trends by Release o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

32.5.2. Defects Density o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

32.5.3. Defect Fix Rate per Release o Defect removal efficiency =  $(1 - (\text{total defects caught by customer/ total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

32.5.4. Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

32.5.5. Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.

Follow the steps below to create a fishbone diagram:

- (13) Write the problem at the head of the fish.
- (14) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
- (15) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
- (16) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement.

Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit.

The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes.

Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

32.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects.

Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes

two major approaches: I) Pareto Analysis and II) Fishbone Analysis

### 32.6.1. Action items, Action Owners, Tracking, Checklist etc.

33. Defect Data • Name of the Person • Types of Testing • Problem Summary • Detailed Description of Defect. • Steps to Reproduce • Life Cycle Phase • Work product where Defect was introduced. • Severity and Priority • Subsystem or Component where the Defect is introduced. • Project Activity occurs when the Defect is introduced. • Identification Method • Type of Defect • Projects and Products in which problems exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References

### 34. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples

- 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached
- 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted
- 3 A minor incident with low impact • A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

### 35. Defect Fix timeline

Severity	Response Time	Resolution Time
1	1	2
2	2	3
3	3	4

### 36. Conclusion

This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

### 37. What is A Defect?

A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

### 38. Defect Management Life Cycle

When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the

projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

### 39. Defect Life Cycle in Detail

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

#### 39.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

#### 39.2. Defect States

#1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.

#2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.

#3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

q) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.

r) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.

s) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.

t) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".

#4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".

#5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".

#6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.

#7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.

#8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.

#9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 39.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual

who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.

- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

#### 40. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

#### 41. Defect Management

##### 41.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

##### 41.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

##### 41.1.2. Why is Defect Tracking Necessary?

- A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- Monitors the bugs that had already been resolved, thereby saving time.
- To ensure that right defects are being worked on.

##### 41.2. Testing stages and defects Responsibilities

SL No. Testing Stage Environment Defect Owner Defect Retested By

Defect Tracking	By	1	Unit Testing (UT)	DEV	DEV Team	Dev Team	2	System Testing (ST)	ST	ST Team	Team	ST Team	ST Team	3	System Integration Testing (SIT)	SIT	SIT Team	SIT Team	SIT Team	4	User Acceptance Testing (UAT)	UAT	UAT Team	UAT Team	Test Manager
-----------------	----	---	-------------------	-----	----------	----------	---	---------------------	----	---------	------	---------	---------	---	----------------------------------	-----	----------	----------	----------	---	-------------------------------	-----	----------	----------	--------------

##### 41.3. Defect Triage

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the “right” defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

##### 41.3.1. Thumb rules of Triaging

- All reported defects have been reviewed
- All accepted defects have been prioritized
- All accepted defects have severity attached
- All rejected defects should have reasonable explanations for the testing team
- Every defect has been assigned to an appropriate owner, individual or team
- Root cause analysis for every accepted defect has been done

##### 41.3.2. How does Defect Triaging work?

The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be

considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

#### 41.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

#### 41.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

##### 41.5.1. Defect Trends by Release

o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

##### 41.5.2. Defects Density

o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

##### 41.5.3. Defect Fix Rate per Release

o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

##### 41.5.4. Defect Aging

o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

##### 41.5.5. Defect Analysis through RCA

o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
- Follow the steps below to create a fishbone diagram:
  - (17) Write the problem at the head of the fish.
  - (18) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
  - (19) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
  - (20) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement. Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the

Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

#### 41.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

#### 41.6.1. Action items, Action Owners, Tracking, Checklist etc.

42. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was resolved and recommendations for testing.
- References

#### 43. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples

- 1 A critical incident with very high impact
- 2 A customer-facing service, like Jira Cloud, is down for all customers
- 3 Confidentiality or privacy is breached
- 4 Customer data loss

2 A major incident with significant impact

3 A customer-facing service is unavailable for a subset of customers

4 Core functionality is significantly impacted

3 A minor incident with low impact

4 A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

#### 44. Defect Fix timeline

Severity Response Time Resolution Time

1	2	3
4	45.	Conclusion

This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in

software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process. 46. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 47.

Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project. 48. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again. 48.1. Defect Workflow The diagram below shows the actual life cycle of a defect. 48.2. Defect States #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle. #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer. #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason. u) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer. v) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer. w) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'. x) Not a Bug: If the defect

does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug". #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed". #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest". #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 48.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 49. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 50. Defect Management

#### 50.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

##### 50.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

##### 50.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

#### 50.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Defect Retested By
1	Unit Testing (UT)	DEV	DEV Team	Dev Team
2	System Testing (ST)	ST	ST Team	ST Team
3	System Integration Testing (SIT)	SIT	SIT Team	SIT Team
4	User Acceptance Testing (UAT)	UAT	UAT Team	UAT Team

#### 50.3. Defect Triage

Defect triage is a process to prioritize defects based on severity, risk, frequency of

occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the “right” defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

50.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

50.3.2. How does Defect Triaging work?

The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

50.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

50.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

50.5.1. Defect Trends by Release

- o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

50.5.2. Defects Density

- o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.
- o Defect Density = Total Defect/Size of the Release

50.5.3. Defect Fix Rate per Release

- o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .
- o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

50.5.4. Defect Aging

- o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

50.5.5. Defect Analysis through RCA

- o RCA

(Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

- o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms. Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish. Follow the steps below to create a fishbone diagram: (21)Write the problem at the head of the fish. (22)Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N] (23)Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N. (24)Extend the causes to secondary, tertiary, and more levels as applicable.
- o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

#### 50.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

#### 50.6.1. Action items, Action Owners, Tracking, Checklist etc.

51. Defect Data
  - Name of the Person
  - Types of Testing
  - Problem Summary
  - Detailed Description of Defect
  - Steps to Reproduce
  - Life Cycle Phase
  - Work product where Defect was introduced.
  - Severity and Priority
  - Subsystem or Component where the Defect is introduced.
  - Project Activity occurs when the Defect is introduced.
  - Identification Method
  - Type of Defect
  - Projects and Products in which problems exist
  - Current Owner
  - Current State of the Report
  - Work product where Defect occurred.
  - Impact on Project
  - Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
  - Dates when various defect lifecycle phases occur.
  - Description of how the defect was resolved and recommendations for testing.
  - References
52. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing

a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted 3 A minor incident with low impact • A minor inconvenience to customers, workaround available In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs Blocker — Highest priority. Indicates that this issue takes precedence over all others. Critical — Indicates that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority. 53. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 54. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process. 55. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 56. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The

number of states that a defect goes through varies from project to project.

**57. Defect Life Cycle in Detail**

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

**57.1. Defect Workflow**

The diagram below shows the actual life cycle of a defect.

**57.2. Defect States**

- #1) **New:** This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) **Assigned:** In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) **Open:** Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.
- y) **Rejected:** If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- z) **Duplicate:** If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.
- aa) **Deferred:** If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.
- bb) **Not a Bug:** If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".
- #4) **Fixed:** When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".
- #5) **Pending Retest:** After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".
- #6) **Retest:** At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.
- #7) **Reopen:** If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.
- #8) **Verified:** If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.
- #9) **Closed:** When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

**57.3. Guidelines for Implementing a Defect Life Cycle**

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

**58. Goals of Defect Management Process (DMP)**

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

**59. Defect Management**

**59.1. Defect Tracking**

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and

business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

59.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

59.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

59.2. Testing stages and defects Responsibilities SL No. Testing Stage Environment Defect Owner Defect Retested By Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team 4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager 59.3. Defect Triage Calls Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

59.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

59.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

59.4. Defect Metrics Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA,

UAT or End-user 59.5. Defect Reporting Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

59.5.1. Defect Trends by Release o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

59.5.2. Defects Density o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

59.5.3. Defect Fix Rate per Release o Defect removal efficiency =  $(1 - (\text{total defects caught by customer/ total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

59.5.4. Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

59.5.5. Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish. Follow the steps below to create a fishbone diagram:

- (25) Write the problem at the head of the fish.
- (26) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
- (27) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
- (28) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

59.6. Defect Prevention Processes & Improvements Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

#1) Review and Inspection: This

method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products. #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system. #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects. #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

59.6.1. Action items, Action Owners, Tracking, Checklist etc.

60. Defect Data • Name of the Person • Types of Testing • Problem Summary • Detailed Description of Defect. • Steps to Reproduce • Life Cycle Phase • Work product where Defect was introduced. • Severity and Priority • Subsystem or Component where the Defect is introduced. • Project Activity occurs when the Defect is introduced. • Identification Method • Type of Defect • Projects and Products in which problems exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References

61. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples

- 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss
- 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted
- 3 A minor incident with low impact • A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

62. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 63. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

64. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

65. Defect Management Life Cycle When a system gives a different output other than the actual

business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

#### 66. Defect Life Cycle in Detail

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

##### 66.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

##### 66.2. Defect States #1)

New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.

##### #2) Assigned:

In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.

##### #3) Open:

Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

##### cc)

##### Rejected:

If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.

##### dd) Duplicate:

If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.

##### ee) Deferred:

If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.

##### ff) Not a Bug:

If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".

##### #4) Fixed:

When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".

##### #5) Pending Retest:

After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".

##### #6) Retest:

At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.

##### #7) Reopen:

If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.

##### #8) Verified:

If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.

##### #9) Closed:

When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields

where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter. 66.3. Guidelines for Implementing a Defect Life Cycle Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

67. Goals of Defect Management Process (DMP) Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

68. Defect Management 68.1. Defect Tracking In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

68.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

68.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

68.2. Testing stages and defects Responsibilities SL No. Testing Stage Environment Defect Owner Defect Retested By Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team 4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager

68.3. Defect Triage Calls Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

68.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

68.3.2. How does Defect Triaging work? The

defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

#### 68.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

#### 68.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

##### 68.5.1. Defect Trends by Release

o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

##### 68.5.2. Defects Density

o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

##### 68.5.3. Defect Fix Rate per Release

o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

##### 68.5.4. Defect Aging

o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

##### 68.5.5. Defect Analysis through RCA

o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
- Follow the steps below to create a fishbone diagram:
  - (29) Write the problem at the head of the fish.
  - (30) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
  - (31) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
  - (32) Extend the

causes to secondary, tertiary, and more levels as applicable.

- o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement. Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

68.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

68.6.1. Action items, Action Owners, Tracking, Checklist etc.

69. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was resolved and recommendations for testing.
- References

70. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples

- 1 A critical incident with very high impact
- A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss
- 2 A major incident with significant impact
- A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted
- 3 A minor incident with low impact
- A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

71. Defect Fix timeline

Severity Response Time Resolution Time

- 1
- 2
- 3
- 4
72. Conclusion

This was all there is to know about the Bug Life Cycle and how the testing

and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

73. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

74. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

75. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

75.1. Defect Workflow The diagram below shows the actual life cycle of a defect.

75.2. Defect States #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle. #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer. #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

gg)

Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer. hh) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer. ii) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

1. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

2. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

3. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

3.1. Defect Workflow The diagram below shows the actual life cycle of a defect.

3.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four

states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

- a) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- b) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.
- c) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.
- d) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".

#4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".

#5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".

#6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.

#7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.

#8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.

#9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 3.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 4. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 5. Defect Management

#### 5.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

##### 5.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

##### 5.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for

evaluation process.

- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

### 5.2. Testing stages and defects

Responsibilities SL No. Testing Stage Environment Defect Owner Defect Retested By Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team 4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager

### 5.3. Defect Triage Calls

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

#### 5.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

#### 5.3.2. How does Defect Triaging work?

The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

### 5.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

### 5.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

#### 5.5.1. Defect Trends by Release

- o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

#### 5.5.2. Defects Density

- o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of

development, divided by the size of the software.

- o Defect Density = Total Defect/Size of the Release
- 5.5.3.Defect Fix Rate per Release o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .
- o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

5.5.4.Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

5.5.5.Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

- o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.

Follow the steps below to create a fishbone diagram:

- (1) Write the problem at the head of the fish.
- (2) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
- (3) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
- (4) Extend the causes to secondary, tertiary, and more levels as applicable.

- o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement

Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

5.6.Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

5.6.1.Action items, Action Owners, Tracking, Checklist etc.

6. Defect Data
  - Name of the Person
  - Types of Testing
  - Problem Summary
  - Detailed Description of Defect.
  - Steps to Reproduce
  - Life Cycle Phase
  - Work product where Defect was introduced.
  - Severity and Priority
  - Subsystem or Component where the Defect is introduced.
  - Project Activity occurs when the Defect is introduced.
  - Identification Method
  - Type of Defect
  - Projects and Products in which problems exist

Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References 7. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted 3 A minor incident with low impact • A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs Blocker — Highest priority. Indicates that this issue takes precedence over all others. Critical — Indicates that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority. 8. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 9. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free.

The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process. 10. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 11. Defect Management Life Cycle

When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally,

the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

### 12. Defect Life Cycle in Detail

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

#### 12.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

#### 12.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.
- e) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- f) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.
- g) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.
- h) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".
- #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".
- #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".
- #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.
- #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.
- #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.
- #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 12.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 13. Goals of Defect Management Process

(DMP) Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

14. Defect Management 14.1. Defect Tracking In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

14.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

14.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

14.2. Testing stages and defects Responsibilities SL No. Testing Stage Environment Defect Owner Defect Retested By Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team 4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager 14.3. Defect Triage Calls Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

14.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

14.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

14.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

14.5. Defect Reporting Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

14.5.1. Defect Trends by Release o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

14.5.2. Defects Density o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

- o Defect Density = Total Defect/Size of the Release

14.5.3. Defect Fix Rate per Release o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

- o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

14.5.4. Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

14.5.5. Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

- o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
- Follow the steps below to create a fishbone diagram:
- (5) Write the problem at the head of the fish.
- (6) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
- (7) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
- (8) Extend the causes to secondary, tertiary, and more levels as applicable.

- o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement.

Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
- 4.

Group Problems Together 5. Add up Scores for Each Group 6. Take Action 14.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

14.6.1. Action items, Action Owners, Tracking, Checklist etc.

15. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was resolved and recommendations for testing.
- References

16. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first?

Severity Description Examples

- 1 A critical incident with very high impact
- A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss
- 2 A major incident with significant impact
- A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted
- 3 A minor incident with low impact
- A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

17. Defect Fix timeline

Severity Response Time Resolution Time 1 2 3 4 18. Conclusion

This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

19. What is A Defect?

A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The

defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 20.

Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project. 21.

Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again. 21.1. Defect Workflow The diagram below shows the actual life cycle of a defect. 21.2.

Defect States #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle. #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer. #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason. i) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer. j) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer. k) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'. l) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug". #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed". #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest". #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the

requirements or not. #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 21.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

## 22. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection

- Minimize the impact
- Resolution of the Defect
- Process improvement

## 23. Defect Management

### 23.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier.

Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

**23.1.1. Objectives of Defect Tracking** A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

**23.1.2. Why is Defect Tracking Necessary?**

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

**23.2. Testing stages and defects Responsibilities**

SL No.	Testing Stage	Environment	Defect Owner	Retested By
1	Unit Testing (UT)	DEV	DEV Team	Dev Team
2	System Testing (ST)	ST	ST Team	Team
3	System Integration Testing (SIT)	SIT	SIT Team	SIT Team
4	User Acceptance Testing (UAT)	UAT	UAT Team	UAT Team

**23.3. Defect Triage**

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should

have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

### 23.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team

Every defect has been assigned to an appropriate owner, individual or team

- o Root cause analysis for every accepted defect has been done

### 23.3.2. How does Defect Triaging work?

The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

### 23.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

### 23.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

#### 23.5.1. Defect Trends by Release

- o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

#### 23.5.2. Defects Density

- o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.
- o Defect Density = Total Defect/Size of the Release

#### 23.5.3. Defect Fix Rate per Release

- o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .
- o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

#### 23.5.4. Defect Aging

- o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

#### 23.5.5. Defect Analysis through RCA

- o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.
- o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its

symptoms. Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish. Follow the steps below to create a fishbone diagram: (9) Write the problem at the head of the fish. (10) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N] (11) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N. (12) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

### 23.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

#### 23.6.1. Action items, Action Owners, Tracking, Checklist etc.

24. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was resolved and recommendations for testing.
- References

25. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first?

Severity Description Examples

- 1 A critical incident with very high impact
- A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss
- 2 A major incident with significant impact
- A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted
- 3 A minor incident with low impact
- A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs Blocker — Highest priority. Indicates that this issue takes precedence over all others. Critical — Indicates that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority.

26. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 27. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

28. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

29. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

30. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

30.1. Defect Workflow The diagram below shows the actual life cycle of a defect.

30.2. Defect States #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this

defect in the later stages of the Defect Life Cycle. #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer. #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason. m) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer. n) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer. o) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'. p) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug". #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed". #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest". #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 30.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 31. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 32. Defect Management

#### 32.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

#### 32.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all

the defects. So that the management may not miss any defect from correction. o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects. o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

### 32.1.2. Why is Defect Tracking Necessary?

o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process. o Monitors the bugs that had already been resolved, thereby saving time. o To ensure that right defects are being worked on.

### 32.2. Testing stages and defects Responsibilities

SL No. Testing Stage Environment Defect Owner Defect Retested By  
Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team 4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager

### 32.3. Defect Triage

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the “right” defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

#### 32.3.1. Thumb rules of Triaging

o All reported defects have been reviewed  
o All accepted defects have been prioritized  
o All accepted defects have severity attached  
o All rejected defects should have reasonable explanations for the testing team  
o Every defect has been assigned to an appropriate owner, individual or team  
o Root cause analysis for every accepted defect has been done

#### 32.3.2. How does Defect Triaging work?

The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

### 32.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

### 32.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect

situation to get feedback from them.

32.5.1. Defect Trends by Release o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

32.5.2. Defects Density o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

32.5.3. Defect Fix Rate per Release o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

32.5.4. Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

32.5.5. Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
- Follow the steps below to create a fishbone diagram:
  - (13) Write the problem at the head of the fish.
  - (14) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
  - (15) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
  - (16) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement.

Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit.

The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes.

Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

32.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects.

Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

32.6.1. Action items,

Action Owners, Tracking, Checklist etc. 33. Defect Data • Name of the Person • Types of Testing • Problem Summary • Detailed Description of Defect. • Steps to Reproduce • Life Cycle Phase • Work product where Defect was introduced. • Severity and Priority • Subsystem or Component where the Defect is introduced. • Project Activity occurs when the Defect is introduced. • Identification Method • Type of Defect • Projects and Products in which problems exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References 34. Defect Priority and Severity Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted 3 A minor incident with low impact • A minor inconvenience to customers, workaround available In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs Blocker — Highest priority. Indicates that this issue takes precedence over all others. Critical — Indicates that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority. 35. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 36. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process. 37. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 38. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but

along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

### 39. Defect Life Cycle in Detail

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

#### 39.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

#### 39.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.
- q) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- r) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.
- s) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.
- t) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".
- #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".
- #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".
- #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.
- #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.
- #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.
- #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

#### 39.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her

responsibility very clearly for better results. • Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily. • The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

#### 40. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

#### 41. Defect Management

##### 41.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

##### 41.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

##### 41.1.2. Why is Defect Tracking Necessary?

- A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- Monitors the bugs that had already been resolved, thereby saving time.
- To ensure that right defects are being worked on.

##### 41.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Defect Retested By
1	Unit Testing (UT)	DEV	DEV Team	Dev Team
2	System Testing (ST)	ST	ST Team	ST Team
3	System Integration Testing (SIT)	SIT	SIT Team	SIT Team
4	User Acceptance Testing (UAT)	UAT	UAT Team	UAT Team

##### 41.3. Defect Triage

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

##### 41.3.1. Thumb rules of Triaging

- All reported defects have been reviewed
- All accepted defects have been prioritized
- All accepted defects have severity attached
- All rejected defects should have reasonable explanations for the testing team
- Every defect has been assigned to an appropriate owner, individual or team
- Root cause analysis for every accepted defect has been done

##### 41.3.2. How does Defect Triaging work?

The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time

sensitivity for resolution • The complexity involved in the resolution • Business impact This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

#### 41.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

#### 41.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

##### 41.5.1. Defect Trends by Release

- o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

##### 41.5.2. Defects Density

- o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.
- o Defect Density = Total Defect/Size of the Release

##### 41.5.3. Defect Fix Rate per Release

- o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .
- o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

##### 41.5.4. Defect Aging

- o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

##### 41.5.5. Defect Analysis through RCA

- o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.
- o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.
- Steps to create a Fishbone Diagram:
  - Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
  - Follow the steps below to create a fishbone diagram:
    - (17)Write the problem at the head of the fish.
    - (18)Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
    - (19)Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
    - (20)Extend the causes to secondary, tertiary, and more levels as applicable.
- o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent

of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

#### 41.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

#### 41.6.1. Action items, Action Owners, Tracking, Checklist etc.

42. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was resolved and recommendations for testing.
- References

#### 43. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first?

Severity Description Examples

- 1 A critical incident with very high impact
- A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss

- 2 A major incident with significant impact
- A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted

- 3 A minor incident with low impact
- A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

#### 44. Defect Fix timeline

Severity Response Time Resolution Time

1	2	3
---	---	---

#### 45. Conclusion

This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed.

Defect Prevention

is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process. 46. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 47.

Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project. 48. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again. 48.1. Defect Workflow The diagram below shows the actual life cycle of a defect. 48.2. Defect States #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle. #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer. #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason. u) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer. v) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer. w) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'. x) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect

gets changed to "Not a Bug". #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed". #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest". #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

#### 48.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

#### 49. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

#### 50. Defect Management

##### 50.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects.

It is an important process in software engineering as Complex and business critical systems have hundreds of defects.

One of the challenging factors is Managing, evaluating and prioritizing these defects.

The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier.

Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

##### 50.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- A defect tracker keeps track of all the defects.

So that the management may not miss any defect from correction.

- Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.

- It saves the time of the engineers thus helping in the fast delivery of the work.

Also, it enhances the efficient work delivered.

##### 50.1.2. Why is Defect Tracking Necessary?

- A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.

- Monitors the bugs that had already been resolved, thereby saving time.

- To ensure that right defects are being worked on.

#### 50.2. Testing stages and defects Responsibilities

SL No. Testing Stage Environment Defect Owner Defect Retested By

Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST

Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team

4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager

50.3. Defect Triage Calls

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more

evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

50.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

50.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

50.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

50.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

50.5.1. Defect Trends by Release

- o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

50.5.2. Defects Density

- o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.
- o Defect Density = Total Defect/Size of the Release

50.5.3. Defect Fix Rate per Release

- o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .
- o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

50.5.4. Defect Aging

- o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

50.5.5. Defect Analysis through RCA

- o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a

Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

- o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.
- Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish. Follow the steps below to create a fishbone diagram:

  - (21) Write the problem at the head of the fish.
  - (22) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
  - (23) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
  - (24) Extend the causes to secondary, tertiary, and more levels as applicable.

- o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement

Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit.

The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes.

Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

#### 50.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects.

Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

##### 50.6.1. Action items, Action Owners, Tracking, Checklist etc.

51. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced
- Severity and Priority
- Subsystem or Component where the Defect is introduced
- Project Activity occurs when the Defect is introduced
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect
- Dates when various defect lifecycle phases occur
- Description of how the defect was resolved and recommendations for testing
- References

52. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder?

Priority, on the other hand,

is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted 3 A minor incident with low impact • A minor inconvenience to customers, workaround available In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs Blocker — Highest priority. Indicates that this issue takes precedence over all others. Critical — Indicates that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority. 53. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 54. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process. 55. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 56. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project. 57. Defect Life

**Cycle in Detail** The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

**57.1. Defect Workflow** The diagram below shows the actual life cycle of a defect.

**57.2. Defect States**

- #1) **New:** This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) **Assigned:** In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) **Open:** Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.
- y) **Rejected:** If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- z) **Duplicate:** If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.
- aa) **Deferred:** If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.
- bb) **Not a Bug:** If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".
- #4) **Fixed:** When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".
- #5) **Pending Retest:** After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".
- #6) **Retest:** At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.
- #7) **Reopen:** If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.
- #8) **Verified:** If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.
- #9) **Closed:** When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

**57.3. Guidelines for Implementing a Defect Life Cycle** Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

**58. Goals of Defect Management Process (DMP)** Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

**59. Defect Management**

**59.1. Defect Tracking** In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing,

evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

59.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

59.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

59.2. Testing stages and defects Responsibilities SL No. Testing Stage Environment Defect Owner Defect Retested By Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team 4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager

59.3. Defect Triage Calls Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

59.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

59.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

59.4. Defect Metrics Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

59.5. Defect Reporting Defect Reporting in software testing is a process in

which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

59.5.1. Defect Trends by Release o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

59.5.2. Defects Density o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

59.5.3. Defect Fix Rate per Release o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

59.5.4. Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

59.5.5. Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish. Follow the steps below to create a fishbone diagram:

- (25) Write the problem at the head of the fish.
- (26) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
- (27) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
- (28) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

59.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and

inspection of all work products. #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system. #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects. #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

59.6.1. Action items, Action Owners, Tracking, Checklist etc.

60. Defect Data • Name of the Person • Types of Testing • Problem Summary • Detailed Description of Defect. • Steps to Reproduce • Life Cycle Phase • Work product where Defect was introduced. • Severity and Priority • Subsystem or Component where the Defect is introduced. • Project Activity occurs when the Defect is introduced. • Identification Method • Type of Defect • Projects and Products in which problems exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References

61. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples

- 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss
- 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted
- 3 A minor incident with low impact • A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

62. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 63. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

64. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

65. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business

requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

#### 66. Defect Life Cycle in Detail

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

##### 66.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

##### 66.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.
- cc) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- dd) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.
- ee) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.
- ff) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".
- #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".
- #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".
- #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.
- #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.
- #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.
- #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase

and Reporter. 66.3. Guidelines for Implementing a Defect Life Cycle Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

67. Goals of Defect Management Process (DMP) Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

68. Defect Management 68.1. Defect Tracking In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

68.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

68.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

68.2. Testing stages and defects Responsibilities SL No. Testing Stage Environment Defect Owner Defect Retested By Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team 4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager 68.3. Defect Triage Calls Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the “right” defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

68.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

68.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes

stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

#### 68.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

#### 68.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

##### 68.5.1. Defect Trends by Release

The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

##### 68.5.2. Defects Density

o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

##### 68.5.3. Defect Fix Rate per Release

o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

##### 68.5.4. Defect Aging

o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

##### 68.5.5. Defect Analysis through RCA

o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
- Follow the steps below to create a fishbone diagram:
  - (29) Write the problem at the head of the fish.
  - (30) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
  - (31) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
  - (32) Extend the causes to secondary, tertiary, and more levels as applicable.
- o Parato charts to do the affinity

analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

68.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

68.6.1. Action items, Action Owners, Tracking, Checklist etc.

69. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was resolved and recommendations for testing.
- References

70. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first?

Severity Description Examples

- 1 A critical incident with very high impact
- A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss

- 2 A major incident with significant impact
- A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted

- 3 A minor incident with low impact
- A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

71. Defect Fix timeline

Severity Response Time Resolution Time

Severity	Response Time	Resolution Time
Blocker	1	2
Critical	2	3
Major	3	4
Minor	4	72.

Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-

free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

73. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

74. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

75. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

75.1. Defect Workflow The diagram below shows the actual life cycle of a defect.

75.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.
- gg)

Rejected: If the defect is not considered a genuine defect by the developer, then it is marked

as "Rejected" by the developer. hh) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer. ii) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

1. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

2. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

3. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

3.1. Defect Workflow The diagram below shows the actual life cycle of a defect.

3.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

a) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer. b) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer. c) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'. d) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug". #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed". #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest". #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

3.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

4. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

5. Defect Management

5.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

5.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

5.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.

To ensure that right defects are being worked on. 5.2.Testing stages and defects

Responsibilities SL No. Testing Stage Environment Defect Owner Defect Retested By Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team 4

User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager 5.3.Defect Triage Calls

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence.

Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized

projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the

team on the "right" defects. Either way, the triage process identifies defects that need

immediate attention, while some may be deferred. The triaging mechanism helps in preparing

a process for testers and developers to fix as many defects as possible by prioritizing them

based on parameters identified and fixed by the team. Ideally, every test cycle should have

regular triage sessions. Frequency can, however, depend on the number of defects identified

with every test. 5.3.1.Thumb rules of Triaging o All reported defects have been reviewed o All

accepted defects have been prioritized o All accepted defects have severity attached o All

rejected defects should have reasonable explanations for the testing team o Every defect has

been assigned to an appropriate owner, individual or team o Root cause analysis for every

accepted defect has been done 5.3.2.How does Defect Triaging work? The defect triage

process involves holding a session / triage call with a triage team, which includes stakeholders

like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business

Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities

and severity level. Priorities correspond to business perspective and severity corresponds to

technicalities. Many times, a few defects may be considered trivial and rejected at this stage.

Accepted defects are prioritized and assigned for resolution. Factors to be considered while

evaluating and prioritizing the defects are: • The validity of the defect • Time sensitivity for

resolution • The complexity involved in the resolution • Business impact This process is not just

about attaching severity and priority to the defects. It also provides all relevant information

required to track, replicate, and fix them. The invalid defects and the basis of their rejection

are also recorded for reference purposes. Root cause analysis for every single defect is

conducted. This analysis forms the basis for an improvement plan which ensures that chances

of getting a similar defect are significantly reduced. 5.4.Defect Metrics Software quality

assurance metrics must be used to track defects and structure the process of their resolution.

Since it is usually not possible to debug every defect in a single sprint, bugs have to be

allocated by priority, severity, testers availability and numerous other parameters. Some

useful defect distribution metrics would be: • Defect distribution by cause • Defect distribution

by feature/functional area • Defect distribution by Severity • Defect distribution by Priority •

Defect distribution by type • Defect distribution by tester (or tester type) – Dev, QA, UAT or

End-user 5.5.Defect Reporting Defect Reporting in software testing is a process in which test

managers / lead prepares and sends the defect report to the management team for feedback

on defect management process and defects' status. Then the management team checks the

defect report and sends feedback or provides further support if needed. Defect reporting helps

to better communicate, track and explain defects in detail. The management board has the

right to know the defect status. They must understand the defect management process to

support you in this project. Therefore, you must report to them the current defect situation to

get feedback from them. 5.5.1.Defect Trends by Release o The Defect Trends report in

Software Testing calculates total number of defects that the QA team has opened and closed

over time. The reports are important for management to understand the overall trend at which

the defects are processed. 5.5.2.Defects Density o Defect density can be defined as the

number of confirmed bugs in a software application or module during the period of

development, divided by the size of the software. o Defect Density = Total Defect/Size of the

Release 5.5.3.Defect Fix Rate per Release o Defect removal efficiency = (1 - (total defects caught by customer/ total number of defects)) x 100. o Residual defect density = (total number of defects found by a customer)/ (Total number of defects including customer found defects) x 100. 5.5.4.Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed). 5.5.5.Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization. o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms. Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish. Follow the steps below to create a fishbone diagram: (1) Write the problem at the head of the fish. (2) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N] (3) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N. (4) Extend the causes to secondary, tertiary, and more levels as applicable. o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

5.6.Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

5.6.1.Action items, Action Owners, Tracking, Checklist etc.

6. Defect Data
  - Name of the Person
  - Types of Testing
  - Problem Summary
  - Detailed Description of Defect.
  - Steps to Reproduce
  - Life Cycle Phase
  - Work product where Defect was introduced.
  - Severity and Priority
  - Subsystem or Component where the Defect is introduced.
  - Project Activity occurs when the Defect is introduced.
  - Identification Method
  - Type of Defect
  - Projects and Products in which problems exist
  - Current Owner
  - Current State of the Report
  - Work product where Defect occurred.
  - Impact

on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.

- Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References

## 7. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted 3 A minor incident with low impact • A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

Blocker — Highest priority. Indicates that this issue takes precedence over all others. Critical — Indicates that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact.

Trivial — Lowest priority.

## 8. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 9.

Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free.

The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

10. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

## 11. Defect Management Life Cycle

When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects.

Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we

need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

### 12. Defect Life Cycle in Detail

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

#### 12.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

#### 12.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.
- e) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- f) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.
- g) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.
- h) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".
- #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".
- #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".
- #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.
- #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.
- #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.
- #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 12.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 13. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection

- Minimize the impact • Resolution of the Defect • Process improvement

**14. Defect Management**

**14.1. Defect Tracking**

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

**14.1.1. Objectives of Defect Tracking**

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

**14.1.2. Why is Defect Tracking Necessary?**

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

**14.2. Testing stages and defects Responsibilities**

SL No.	Testing Stage	Environment	Defect Owner	Defect Retested By
--------	---------------	-------------	--------------	--------------------

Defect Tracking	By 1 Unit Testing (UT)	DEV	DEV Team	Dev Team	2 System Testing (ST)	ST	ST Team	ST Team	ST Team	3 System Integration Testing (SIT)	SIT	SIT Team	SIT Team	SIT Team
-----------------	------------------------	-----	----------	----------	-----------------------	----	---------	---------	---------	------------------------------------	-----	----------	----------	----------

4 User Acceptance Testing (UAT)	UAT	UAT Team	UAT Team	Test Manager
---------------------------------	-----	----------	----------	--------------

**14.3. Defect Triage**

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

**14.3.1. Thumb rules of Triaging**

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

**14.3.2. How does Defect Triaging work?**

The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

**14.4. Defect Metrics**

Software quality assurance metrics must be used to track defects and structure the process of

their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

14.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

14.5.1. Defect Trends by Release

o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

14.5.2. Defects Density

o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

14.5.3. Defect Fix Rate per Release

o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

14.5.4. Defect Aging

o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

14.5.5. Defect Analysis through RCA

o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram

A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish. Follow the steps below to create a fishbone diagram:

- (5) Write the problem at the head of the fish.
- (6) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
- (7) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
- (8) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts

to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement

Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit.

The 80/20 Rule

Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes.

Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff.

The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

14.6. Defect

**Prevention Processes & Improvements** Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches:
  - I) Pareto Analysis
  - II) Fishbone Analysis

14.6.1. Action items, Action Owners, Tracking, Checklist etc.

15. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was resolved and recommendations for testing.
- References

16. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first?

Severity Description Examples

- 1 A critical incident with very high impact
- A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss
- 2 A major incident with significant impact
- A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted
- 3 A minor incident with low impact
- A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

17. Defect Fix timeline

Severity	Response Time	Resolution Time
1	1	2
2	2	3
3	4	8

18. Conclusion

This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

19. What is A Defect?

A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an

application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 20.

**Defect Management Life Cycle** When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project. 21.

**Defect Life Cycle in Detail** The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

**21.1. Defect Workflow** The diagram below shows the actual life cycle of a defect. 21.2. **Defect States** #1) **New:** This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle. #2) **Assigned:** In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer. #3) **Open:** Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason. i) **Rejected:** If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer. j) **Duplicate:** If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer. k) **Deferred:** If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'. l) **Not a Bug:** If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug". #4) **Fixed:** When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed". #5) **Pending Retest:** After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest". #6) **Retest:** At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. #7) **Reopen:** If any issue persists in the defect, then it will be assigned to

the developer again for testing and the status of the defect gets changed to 'Reopen'. #8)

**Verified:** If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) **Closed:** When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields

longer, then the tester changes the status of the defect to ‘Closed’. The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter. 21.3. Guidelines for Implementing a Defect Life Cycle Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

22. Goals of Defect Management Process (DMP) Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

23. Defect

**Management 23.1. Defect Tracking** In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier.

Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

23.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction. o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects. o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

23.1.2. Why is Defect Tracking

Necessary? o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process. o Monitors the bugs that had already been resolved, thereby

saving time. o To ensure that right defects are being worked on. 23.2. Testing stages and defects Responsibilities SL No. Testing Stage Environment Defect Owner Defect Retested By

Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team 4 User Acceptance Testing (UAT) UAT UAT UAT UAT UAT UAT UAT UAT UAT

4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager 23.3. Defect Triage Calls Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more

occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help

large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the “right” defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in

need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should

them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects

identified with every test.

### 23.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

### 23.3.2. How does Defect Triaging work?

The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

### 23.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

### 23.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

#### 23.5.1. Defect Trends by Release

- o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

#### 23.5.2. Defects Density

- o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.
- o Defect Density = Total Defect/Size of the Release

#### 23.5.3. Defect Fix Rate per Release

- o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .
- o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

#### 23.5.4. Defect Aging

- o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

#### 23.5.5. Defect Analysis through RCA

- o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.
- o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a

fish with the problem forming the head of fish and causes forming the spine and bones of the fish. Follow the steps below to create a fishbone diagram: (9) Write the problem at the head of the fish. (10) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N] (11) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N. (12) Extend the causes to secondary, tertiary, and more levels as applicable.

o Pareto charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement

Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes.

Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

### 23.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

#### 23.6.1. Action items, Action Owners, Tracking, Checklist etc.

24. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was resolved and recommendations for testing.
- References

25. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first?

Severity Description Examples

- 1 A critical incident with very high impact
- A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss

- 2 A major incident with significant impact
- A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted

- 3 A minor incident with low impact
- A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial

Bugs Blocker —

Highest priority. Indicates that this issue takes precedence over all others. Critical — Indicates that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority.

26. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4

27. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

28. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

29. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

30. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

30.1. Defect Workflow The diagram below shows the actual life cycle of a defect.

30.2. Defect States #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.

#2) Assigned: In this stage, a newly created

defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer. #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason. m)

Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer. n) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer. o) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'. p) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug". #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".

#5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest". #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8)

Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

30.3. Guidelines for Implementing a Defect Life Cycle Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

31. Goals of Defect Management Process (DMP) Given below are the various goals of this process:

- Prevent the Defect
- Early Detection

- Minimize the impact
- Resolution of the Defect
- Process improvement

32. Defect Management 32.1. Defect Tracking In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier.

Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity

- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

32.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these

tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects. o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

### 32.1.2. Why is Defect Tracking Necessary?

o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.

o Monitors the bugs that had already been resolved, thereby saving time.

o To ensure that right defects are being worked on.

### 32.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Defect Retested By
1	Unit Testing (UT)	DEV	DEV Team	Dev Team
2	System Testing (ST)	ST	ST	Team
3	System Integration Testing (SIT)	SIT	SIT	Team
4	User Acceptance Testing (UAT)	UAT	UAT Team	UAT Team
				Test Manager

### 32.3. Defect Triage Calls

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

#### 32.3.1. Thumb rules of Triaging

o All reported defects have been reviewed

o All accepted defects have been prioritized

o All accepted defects have severity attached

o All rejected defects should have reasonable explanations for the testing team

o Every defect has been assigned to an appropriate owner, individual or team

o Root cause analysis for every accepted defect has been done

#### 32.3.2. How does Defect Triaging work?

The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

### 32.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

### 32.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

#### 32.5.1. Defect Trends by Release

o The Defect Trends

report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

32.5.2. Defect Density o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

32.5.3. Defect Fix Rate per Release o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

32.5.4. Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

32.5.5. Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
- Follow the steps below to create a fishbone diagram:
  - (13) Write the problem at the head of the fish.
  - (14) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
  - (15) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
  - (16) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement.

Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit.

The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes.

Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

32.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects.

Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

32.6.1. Action items, Action Owners, Tracking, Checklist etc.

33. Defect Data

- Name of the Person
- Types of

Testing • Problem Summary • Detailed Description of Defect. • Steps to Reproduce • Life Cycle Phase • Work product where Defect was introduced. • Severity and Priority • Subsystem or Component where the Defect is introduced. • Project Activity occurs when the Defect is introduced. • Identification Method • Type of Defect • Projects and Products in which problems exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References

34. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples

- 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss

2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers

• Core functionality is significantly impacted

3 A minor incident with low impact • A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

35. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4

36. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

37. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

38. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if

the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

### 39. Defect Life Cycle in Detail

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

#### 39.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

#### 39.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.
- q) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- r) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.
- s) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.
- t) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".
- #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".
- #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".
- #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.
- #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.
- #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.
- #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

#### 39.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a

defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.

- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

40. Goals of Defect Management Process (DMP) Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

## 41. Defect Management

### 41.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

#### 41.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

#### 41.1.2. Why is Defect Tracking Necessary?

- A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- Monitors the bugs that had already been resolved, thereby saving time.
- To ensure that right defects are being worked on.

#### 41.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Defect Retested By
1	Unit Testing (UT)	DEV	DEV Team	Dev Team
2	System Testing (ST)	ST	ST Team	Team ST
3	System Integration Testing (SIT)	SIT	SIT Team	SIT Team
4	User Acceptance Testing (UAT)	UAT	UAT Team	UAT Team

#### 41.3. Defect Triage

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the “right” defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

#### 41.3.1. Thumb rules of Triaging

- All reported defects have been reviewed
- All accepted defects have been prioritized
- All accepted defects have severity attached
- All rejected defects should have reasonable explanations for the testing team
- Every defect has been assigned to an appropriate owner, individual or team
- Root cause analysis for every accepted defect has been done

#### 41.3.2. How does Defect Triaging work?

The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This

process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

#### 41.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

#### 41.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

##### 41.5.1. Defect Trends by Release

- The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

##### 41.5.2. Defects Density

- Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.
- Defect Density = Total Defect/Size of the Release

##### 41.5.3. Defect Fix Rate per Release

- Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .
- Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

##### 41.5.4. Defect Aging

- Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

##### 41.5.5. Defect Analysis through RCA

- RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.
- Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.
- Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish. Follow the steps below to create a fishbone diagram:
  - (17) Write the problem at the head of the fish.
  - (18) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
  - (19) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
  - (20) Extend the causes to secondary, tertiary, and more levels as applicable.
- Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of

problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

#### 41.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

#### 41.6.1. Action items, Action Owners, Tracking, Checklist etc.

42. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was resolved and recommendations for testing.
- References

#### 43. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first?

Severity Description Examples

- 1 A critical incident with very high impact
- 2 A customer-facing service, like Jira Cloud, is down for all customers
- 3 Confidentiality or privacy is breached
- 4 Customer data loss

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

#### 44. Defect Fix timeline

Severity Response Time Resolution Time

1	2	3
---	---	---

#### 45. Conclusion

This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost

effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

46. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

47. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

48. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

48.1. Defect Workflow The diagram below shows the actual life cycle of a defect.

48.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.
- u) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- v) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.
- w) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.
- x) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".
- #4) Fixed: When the developer finishes the task of fixing a

defect by making the required changes then they can mark the status of the defect as "Fixed". #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest". #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 48.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 49. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 50. Defect Management

#### 50.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

##### 50.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

##### 50.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

#### 50.2. Testing stages and defects Responsibilities

SL No. Testing Stage Environment Defect Owner Defect Retested By Defect Tracking By

1	Unit Testing (UT)	DEV DEV Team	Dev Team	2	System Testing (ST)	ST ST Team	ST Team	ST Team	3	System Integration Testing (SIT)	SIT SIT Team	SIT Team	SIT Team
---	-------------------	--------------	----------	---	---------------------	------------	---------	---------	---	----------------------------------	--------------	----------	----------

#### 50.3. Defect Triage Calls

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for

large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the “right” defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

50.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

50.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

50.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

50.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

50.5.1. Defect Trends by Release

- o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

50.5.2. Defects Density

- o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.
- o 
$$\text{Defect Density} = \frac{\text{Total Defect}}{\text{Size of the Release}}$$

50.5.3. Defect Fix Rate per Release

- o 
$$\text{Defect removal efficiency} = (1 - \frac{\text{total defects caught by customer}}{\text{total number of defects}}) \times 100$$
- o 
$$\text{Residual defect density} = \frac{\text{total number of defects found by a customer}}{\text{(Total number of defects including customer found defects)}} \times 100$$

50.5.4. Defect Aging

- o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

50.5.5. Defect Analysis through RCA

- o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and

quality of the deliverables and the processes, not only at the team level but also across the organization.

- o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
- Follow the steps below to create a fishbone diagram:
- (21) Write the problem at the head of the fish.
- (22) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
- (23) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
- (24) Extend the causes to secondary, tertiary, and more levels as applicable.

- o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement.

Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit.

The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes.

Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

#### 50.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects.

Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

#### 50.6.1. Action items, Action Owners, Tracking, Checklist etc.

51. Defect Data
  - Name of the Person
  - Types of Testing
  - Problem Summary
  - Detailed Description of Defect
  - Steps to Reproduce
  - Life Cycle Phase
  - Work product where Defect was introduced
  - Severity and Priority
  - Subsystem or Component where the Defect is introduced
  - Project Activity occurs when the Defect is introduced
  - Identification Method
  - Type of Defect
  - Projects and Products in which problems exist
  - Current Owner
  - Current State of the Report
  - Work product where Defect occurred
  - Impact on Project
  - Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect
  - Dates when various defect lifecycle phases occur
  - Description of how the defect was resolved and recommendations for testing
  - References
52. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to

be fixed first? Severity Description Examples 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted 3 A minor incident with low impact • A minor inconvenience to customers, workaround available In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs Blocker — Highest priority. Indicates that this issue takes precedence over all others. Critical — Indicates that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority. 53. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 54. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process. 55. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 56. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project. 57. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering

the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

**57.1. Defect Workflow** The diagram below shows the actual life cycle of a defect.

**57.2. Defect States #1)** New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.

#2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.

#3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

y) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.

z) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.

aa) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.

bb) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".

#4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".

#5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".

#6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.

#7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.

#8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.

#9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

**57.3. Guidelines for Implementing a Defect Life Cycle** Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

**58. Goals of Defect Management Process (DMP)** Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

**59. Defect Management**

**59.1. Defect Tracking** In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of

time and to effectively manage them, defect tracking system is used to make the job easier.

Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity

- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

59.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

59.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.

- o To ensure that right defects are being worked on.

59.2. Testing stages and defects Responsibilities SL No. Testing Stage Environment Defect Owner Defect Retested By

Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team 4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager

59.3. Defect Triage Calls Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

59.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

59.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

59.4. Defect Metrics Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

59.5. Defect Reporting Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for

feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

59.5.1. Defect Trends by Release o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

59.5.2. Defects Density o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

59.5.3. Defect Fix Rate per Release o Defect removal efficiency =  $(1 - (\text{total defects caught by customer/ total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

59.5.4. Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

59.5.5. Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
- Follow the steps below to create a fishbone diagram:
  - (25) Write the problem at the head of the fish.
  - (26) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
  - (27) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
  - (28) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

59.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's

mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system. #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects. #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

59.6.1. Action items, Action Owners, Tracking, Checklist etc.

60. Defect Data • Name of the Person • Types of Testing • Problem Summary • Detailed Description of Defect. • Steps to Reproduce • Life Cycle Phase • Work product where Defect was introduced. • Severity and Priority • Subsystem or Component where the Defect is introduced. • Project Activity occurs when the Defect is introduced. • Identification Method • Type of Defect • Projects and Products in which problems exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References

61. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples

1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss

2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted

3 A minor incident with low impact • A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

Blocker — Highest priority. Indicates that this issue takes precedence over all others. Critical — Indicates that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority.

62. Defect Fix timeline

Severity Response Time Resolution Time 1 2 3 4 63. Conclusion

This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process

Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

64. What is A Defect?

A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

65. Defect Management Life Cycle

When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing

team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

### 66. Defect Life Cycle in Detail

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

#### 66.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

#### 66.2. Defect States

#1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.

#2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.

#3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

cc)

Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.

dd) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.

ee) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.

ff) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".

#4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".

#5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".

#6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.

#7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.

#8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.

#9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

#### 66.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines

can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

67. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

68. Defect Management

68.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

68.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

68.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

68.2. Testing stages and defects Responsibilities

SL No. Testing Stage Environment Defect Owner Defect Retested By Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team 4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager

68.3. Defect Triage Calls

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

68.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

68.3.2. How does Defect Triaging work?

The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and

Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

#### 68.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

#### 68.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

##### 68.5.1. Defect Trends by Release

- The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

##### 68.5.2. Defects Density

- Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.
- Defect Density = Total Defect/Size of the Release

##### 68.5.3. Defect Fix Rate per Release

- Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .
- Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

##### 68.5.4. Defect Aging

- Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

##### 68.5.5. Defect Analysis through RCA

- RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.
- Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
- Follow the steps below to create a fishbone diagram:
  - (29) Write the problem at the head of the fish.
  - (30) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
  - (31) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
  - (32) Extend the causes to secondary, tertiary, and more levels as applicable.

##### ◦ Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their

cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement. Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

#### 6.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

#### 6.6.1. Action items, Action Owners, Tracking, Checklist etc.

6.9. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was resolved and recommendations for testing.
- References

#### 70. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first?

Severity Description Examples

- 1 A critical incident with very high impact
- A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss

- 2 A major incident with significant impact
- A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted

- 3 A minor incident with low impact
- A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

#### 71. Defect Fix timeline

Severity Response Time Resolution Time

Severity	Response Time	Resolution Time
Blocker	1	2
Critical	2	3
Major	3	4
Minor	4	5
Trivial	5	6

#### 72. Conclusion

This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software

development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

73. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

74. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

75. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

75.1. Defect Workflow The diagram below shows the actual life cycle of a defect.

75.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.
- gg) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- hh) Duplicate: If the developer finds the defect as same as

any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer. ii) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process. 1. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 2. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project. 3. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again. 3.1. Defect Workflow The diagram below shows the actual life cycle of a defect. 3.2. Defect States #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle. #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer. #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason. a) Rejected: If the defect is not considered a genuine defect by the developer, then it is

marked as "Rejected" by the developer. b) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer. c) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'. d) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug". #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed". #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest". #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 3.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 4. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 5. Defect Management

#### 5.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

##### 5.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

##### 5.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

#### 5.2. Testing stages and defects

Responsibilities SL No. Testing Stage Environment Defect Owner Defect Retested By Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team 4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager 5.3.Defect Triage Calls Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test. 5.3.1.Thumb rules of Triaging o All reported defects have been reviewed o All accepted defects have been prioritized o All accepted defects have severity attached o All rejected defects should have reasonable explanations for the testing team o Every defect has been assigned to an appropriate owner, individual or team o Root cause analysis for every accepted defect has been done 5.3.2.How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are: • The validity of the defect • Time sensitivity for resolution • The complexity involved in the resolution • Business impact This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced. 5.4.Defect Metrics Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be: • Defect distribution by cause • Defect distribution by feature/functional area • Defect distribution by Severity • Defect distribution by Priority • Defect distribution by type • Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user 5.5.Defect Reporting Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them. 5.5.1.Defect Trends by Release o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed. 5.5.2.Defects Density o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software. o Defect Density = Total Defect/Size of the Release 5.5.3.Defect Fix Rate per Release o Defect removal efficiency = (1 - (total defects

caught by customer/ total number of defects)) x 100. o Residual defect density = (total number of defects found by a customer)/ (Total number of defects including customer found defects) x 100.

5.5.4. Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

5.5.5. Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.

Follow the steps below to create a fishbone diagram:

- (1) Write the problem at the head of the fish.
- (2) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
- (3) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
- (4) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement

Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit.

The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes.

Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

5.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects.

Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

5.6.1. Action items, Action Owners, Tracking, Checklist etc.

6. Defect Data
  - Name of the Person
  - Types of Testing
  - Problem Summary
  - Detailed Description of Defect.
  - Steps to Reproduce
  - Life Cycle Phase
  - Work product where Defect was introduced.
  - Severity and Priority
  - Subsystem or Component where the Defect is introduced.
  - Project Activity occurs when the Defect is introduced.
  - Identification Method
  - Type of Defect
  - Projects and Products in which problems exist
  - Current Owner
  - Current State of the Report
  - Work product where Defect occurred.
  - Impact on Project
  - Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.

- Dates when various defect lifecycle phases occur.
  - Description of how the defect was resolved and recommendations for testing.
  - References
7. Defect Priority and Severity
- Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples
- 1 A critical incident with very high impact
  - A customer-facing service, like Jira Cloud, is down for all customers
  - Confidentiality or privacy is breached
  - Customer data loss
  - 2 A major incident with significant impact
  - A customer-facing service is unavailable for a subset of customers
  - Core functionality is significantly impacted
  - 3 A minor incident with low impact
  - A minor inconvenience to customers, workaround available
- In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs
- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
  - Critical — Indicates that this issue is causing a problem and requires urgent attention.
  - Major — Indicates that this issue has a significant impact.
  - Minor — Indicates that this issue has a relatively minor impact.
  - Trivial — Lowest priority.
8. Defect Fix timeline
- | Severity | Time Resolution |
|----------|-----------------|
| 1        | 1               |
| 2        | 2               |
| 3        | 3               |
9. Conclusion
- This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process
- Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.
10. What is A Defect?
- A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.
11. Defect Management Life Cycle
- When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and

standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

### 12. Defect Life Cycle in Detail

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

#### 12.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

#### 12.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.
- e) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- f) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.
- g) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.
- h) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".
- #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".
- #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".
- #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.
- #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.
- #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.
- #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 12.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 13. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 14. Defect

**Management 14.1. Defect Tracking** In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

**14.1.1. Objectives of Defect Tracking** A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

**14.1.2. Why is Defect Tracking Necessary?**

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

**14.2. Testing stages and defects Responsibilities** SL No. Testing Stage Environment Defect Owner Defect Retested By

Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST

Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team

4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager

**14.3. Defect Triage Calls** Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

**14.3.1. Thumb rules of Triaging**

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

**14.3.2. How does Defect Triaging work?** The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

**14.4. Defect Metrics** Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs

have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

14.5. Defect Reporting Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

14.5.1. Defect Trends by Release o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

14.5.2. Defects Density o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

14.5.3. Defect Fix Rate per Release o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

14.5.4. Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

14.5.5. Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish. Follow the steps below to create a fishbone diagram:

- (5) Write the problem at the head of the fish.
- (6) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
- (7) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
- (8) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement

Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

14.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process

of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

14.6.1. Action items, Action Owners, Tracking, Checklist etc.

15. Defect Data • Name of the Person • Types of Testing • Problem Summary • Detailed Description of Defect. • Steps to Reproduce • Life Cycle Phase • Work product where Defect was introduced. • Severity and Priority • Subsystem or Component where the Defect is introduced. • Project Activity occurs when the Defect is introduced. • Identification Method • Type of Defect • Projects and Products in which problems exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References

16. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first?

Severity Description Examples

- 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss
- 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted
- 3 A minor incident with low impact • A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

17. Defect Fix timeline

Severity	Response Time	Resolution Time
1	2	3

18. Conclusion

This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

19. What is A Defect?

A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the

responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 20.

**Defect Management Life Cycle** When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project. 21. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again. 21.1. Defect Workflow The diagram below shows the actual life cycle of a defect. 21.2. Defect States #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle. #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer. #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason. i) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer. j) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer. k) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'. l) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug". #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed". #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest". #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8)

Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

21.3. Guidelines for Implementing a Defect Life Cycle Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

22. Goals of Defect Management Process (DMP) Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

23. Defect Management

23.1. Defect Tracking In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

23.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

23.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

23.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Defect Retested By
1	Unit Testing (UT)	DEV	DEV Team	Dev Team
2	System Testing (ST)	ST	ST Team	ST Team
3	System Integration Testing (SIT)	SIT	SIT Team	SIT Team
4	User Acceptance Testing (UAT)	UAT	UAT Team	UAT Team

23.3. Defect Triage Calls

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

23.3.1. Thumb rules of Triaging

- o All reported defects have been

reviewed o All accepted defects have been prioritized o All accepted defects have severity attached o All rejected defects should have reasonable explanations for the testing team o Every defect has been assigned to an appropriate owner, individual or team o Root cause analysis for every accepted defect has been done 23.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

23.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

23.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

23.5.1. Defect Trends by Release

- o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

23.5.2. Defects Density

- o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.
- o Defect Density = Total Defect/Size of the Release

23.5.3. Defect Fix Rate per Release

- o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100.$
- o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100.$

23.5.4. Defect Aging

- o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

23.5.5. Defect Analysis through RCA

- o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.
- o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the

fish. Follow the steps below to create a fishbone diagram: (9) Write the problem at the head of the fish. (10) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N] (11) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N. (12) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

### 23.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

#### 23.6.1. Action items, Action Owners, Tracking, Checklist etc.

24. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was resolved and recommendations for testing.
- References

25. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples

- 1 A critical incident with very high impact
- A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss
- 2 A major incident with significant impact
- A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted
- 3 A minor incident with low impact
- A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

Blocker — Highest priority. Indicates that this issue takes precedence over all others.

Critical — Indicates

that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority.

26. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 27. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

28. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

29. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

30. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

30.1. Defect Workflow The diagram below shows the actual life cycle of a defect.

30.2. Defect States #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle. #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA

or test lead of the testing team to a developer. #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason. m) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer. n) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer. o) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'. p) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug". #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed". #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest". #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 30.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 31. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 32. Defect Management

#### 32.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

#### 32.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to

avoid further defects. o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

### 32.1.2. Why is Defect Tracking Necessary?

o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.

o Monitors the bugs that had already been resolved, thereby saving time.

o To ensure that right defects are being worked on.

### 32.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Defect Retested By
1	Unit Testing (UT)	DEV	DEV Team	Dev Team
2	System Testing (ST)	ST	ST Team	ST Team
3	System Integration Testing (SIT)	SIT	SIT Team	SIT Team
4	User Acceptance Testing (UAT)	UAT	UAT Team	UAT Team

### 32.3. Defect Triage Calls

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

#### 32.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

#### 32.3.2. How does Defect Triaging work?

The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

### 32.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

### 32.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

#### 32.5.1. Defect Trends by Release

- o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and

closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

32.5.2. Defect Density o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

32.5.3. Defect Fix Rate per Release o Defect removal efficiency = (1 - (total defects caught by customer/ total number of defects)) x 100.

o Residual defect density = (total number of defects found by a customer)/ (Total number of defects including customer found defects) x 100.

32.5.4. Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

32.5.5. Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
- Follow the steps below to create a fishbone diagram:
  - (13) Write the problem at the head of the fish.
  - (14) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
  - (15) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
  - (16) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement

Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit.

The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes.

Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

32.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

32.6.1. Action items, Action Owners, Tracking, Checklist etc.

33. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect
- Steps to Reproduce
- Life Cycle

Phase • Work product where Defect was introduced. • Severity and Priority • Subsystem or Component where the Defect is introduced. • Project Activity occurs when the Defect is introduced. • Identification Method • Type of Defect • Projects and Products in which problems exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References

34. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples

- 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss
- 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted
- 3 A minor incident with low impact • A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

35. Defect Fix timeline

Severity Response Time Resolution Time 1 2 3

4 36. Conclusion

This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process

Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

37. What is A Defect?

A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

38. Defect Management Life Cycle

When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the

status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

### 39. Defect Life Cycle in Detail

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

#### 39.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

#### 39.2. Defect States

#1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.

#2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.

#3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

q) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.

r) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.

s) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.

t) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".

#4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".

#5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".

#6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.

#7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.

#8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.

#9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

#### 39.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the

status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily. • The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

40. Goals of Defect Management Process (DMP) Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

41. Defect Management

41.1. Defect Tracking In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

41.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

41.1.2. Why is Defect Tracking Necessary?

- A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- Monitors the bugs that had already been resolved, thereby saving time.
- To ensure that right defects are being worked on.

41.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Defect Retested By
Defect Tracking	By 1	Unit Testing (UT)	DEV	DEV Team
Team	ST	Team	Dev Team	2 System Testing (ST)
ST	Team	ST	Team	3 System Integration Testing (SIT)
Team	SIT	SIT	Team	SIT Team
4	User Acceptance Testing (UAT)	UAT	UAT Team	UAT Team
			Test Manager	

41.3. Defect Triage

Calls

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

41.3.1. Thumb rules of Triaging

- All reported defects have been reviewed
- All accepted defects have been prioritized
- All accepted defects have severity attached
- All rejected defects should have reasonable explanations for the testing team
- Every defect has been assigned to an appropriate owner, individual or team
- Root cause analysis for every accepted defect has been done

41.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all

relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

#### 41.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

#### 41.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

##### 41.5.1. Defect Trends by Release

o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

##### 41.5.2. Defects Density

o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

##### 41.5.3. Defect Fix Rate per Release

o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

##### 41.5.4. Defect Aging

o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

##### 41.5.5. Defect Analysis through RCA

o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
- Follow the steps below to create a fishbone diagram:
  - (17) Write the problem at the head of the fish.
  - (18) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
  - (19) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
  - (20) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement.

Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit.

The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem

areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

#### 41.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

#### 41.6.1. Action items, Action Owners, Tracking, Checklist etc.

42. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was resolved and recommendations for testing.
- References

43. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first?

Severity Description Examples

- 1 A critical incident with very high impact
- A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss

- 2 A major incident with significant impact
- A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted

- 3 A minor incident with low impact
- A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

44. Defect Fix timeline

Severity Response Time Resolution Time

Severity	Response Time	Resolution Time
Blocker	Within 24 hours	Within 72 hours
Critical	Within 48 hours	Within 1 week
Major	Within 7 days	Within 2 weeks
Minor	Within 14 days	Within 1 month
Trivial	Within 30 days	Within 3 months

45. Conclusion

This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The

organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process. 46. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 47.

Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project. 48. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again. 48.1. Defect Workflow The diagram below shows the actual life cycle of a defect. 48.2. Defect States #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle. #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer. #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason. u) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer. v) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer. w) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'. x) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug". #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".

#5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest". #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

48.3. Guidelines for Implementing a Defect Life Cycle Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

49. Goals of Defect Management Process (DMP) Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

50. Defect Management 50.1. Defect Tracking In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

50.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

50.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

50.2. Testing stages and defects Responsibilities SL No. Testing Stage Environment Defect Owner Defect Retested By Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team 4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager 50.3. Defect Triage Calls Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help

focus the team on the “right” defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

50.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team

Every defect has been assigned to an appropriate owner, individual or team

- o Root cause analysis for every accepted defect has been done

50.3.2. How does Defect Triaging work?

The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

50.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

50.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

50.5.1. Defect Trends by Release

- o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

50.5.2. Defects Density

- o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.
- o Defect Density = Total Defect/Size of the Release

50.5.3. Defect Fix Rate per Release

- o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100.$
- o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100.$

50.5.4. Defect Aging

- o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

50.5.5. Defect Analysis through RCA

- o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the

organization.

- o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms. Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish. Follow the steps below to create a fishbone diagram: (21)Write the problem at the head of the fish. (22)Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N] (23)Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N. (24)Extend the causes to secondary, tertiary, and more levels as applicable.
- o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

#### 50.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

#### 50.6.1. Action items, Action Owners, Tracking, Checklist etc.

51. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was resolved and recommendations for testing.
- References

#### 52. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples

- 1 A critical incident with very high impact
- A

customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted 3 A minor incident with low impact • A minor inconvenience to customers, workaround available In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs Blocker — Highest priority. Indicates that this issue takes precedence over all others. Critical — Indicates that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority. 53. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 54. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process. 55. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 56. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project. 57. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester

and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

57.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

57.2. Defect States

#1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.

#2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.

#3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

y) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.

z) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.

aa) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.

bb) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".

#4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".

#5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".

#6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.

#7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.

#8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.

#9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

57.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

58. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

59. Defect Management

59.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier.

Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

59.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

59.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

59.2. Testing stages and defects Responsibilities SL No. Testing Stage Environment Defect Owner Defect Retested By Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team 4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager 59.3. Defect Triage Calls Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

59.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

59.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

59.4. Defect Metrics Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

59.5. Defect Reporting Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team

checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

59.5.1. Defect Trends by Release o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

59.5.2. Defects Density o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

59.5.3. Defect Fix Rate per Release o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

59.5.4. Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

59.5.5. Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish. Follow the steps below to create a fishbone diagram:

- (25) Write the problem at the head of the fish.
- (26) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
- (27) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
- (28) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

59.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea

regarding the correctness and/or the look and-feel of the system. #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects. #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

59.6.1. Action items, Action Owners, Tracking, Checklist etc.

60. Defect Data • Name of the Person • Types of Testing • Problem Summary • Detailed Description of Defect. • Steps to Reproduce • Life Cycle Phase • Work product where Defect was introduced. • Severity and Priority • Subsystem or Component where the Defect is introduced. • Project Activity occurs when the Defect is introduced. • Identification Method • Type of Defect • Projects and Products in which problems exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References

61. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples

1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss

2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted

3 A minor incident with low impact • A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

Blocker — Highest priority. Indicates that this issue takes precedence over all others. Critical — Indicates that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority.

62. Defect Fix timeline

Severity Response Time Resolution Time 1 2 3 4 63. Conclusion

This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

64. What is A Defect?

A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

65. Defect Management Life Cycle

When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is

different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

66. Defect Life Cycle in Detail

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

66.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

66.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.
- cc) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- dd) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.
- ee) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.
- ff) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".
- #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".
- #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".
- #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.
- #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.
- #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.
- #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

66.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is

very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).

- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 67. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 68. Defect Management

#### 68.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

#### 68.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

#### 68.1.2. Why is Defect Tracking Necessary?

- A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- Monitors the bugs that had already been resolved, thereby saving time.
- To ensure that right defects are being worked on.

#### 68.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Defect Retested By
1	Unit Testing (UT)	DEV	DEV Team	Dev Team
2	System Testing (ST)	ST	ST Team	ST Team
3	System Integration Testing (SIT)	SIT	SIT Team	SIT Team
4	User Acceptance Testing (UAT)	UAT	UAT Team	UAT Team
			Test Manager	

#### 68.3. Defect Triage

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

#### 68.3.1. Thumb rules of Triaging

- All reported defects have been reviewed
- All accepted defects have been prioritized
- All accepted defects have severity attached
- All rejected defects should have reasonable explanations for the testing team
- Every defect has been assigned to an appropriate owner, individual or team
- Root cause analysis for every accepted defect has been done

#### 68.3.2. How does Defect Triaging work?

The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach

priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

#### 68.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

#### 68.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

##### 68.5.1. Defect Trends by Release

o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

##### 68.5.2. Defects Density

o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

##### 68.5.3. Defect Fix Rate per Release

o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

##### 68.5.4. Defect Aging

o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

##### 68.5.5. Defect Analysis through RCA

o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
- Follow the steps below to create a fishbone diagram:
  - (29) Write the problem at the head of the fish.
  - (30) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
  - (31) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
  - (32) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe

the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

### 6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

#### 6.1. Action items, Action Owners, Tracking, Checklist etc.

#### 6.9. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was resolved and recommendations for testing.
- References

#### 7.0. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first?

Severity Description Examples

- 1 A critical incident with very high impact
- A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss

- 2 A major incident with significant impact
- A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted

- 3 A minor incident with low impact
- A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

#### 7.1. Defect Fix timeline

Severity Response Time Resolution Time

1	2	3
4	7	2

#### 7.2. Conclusion

This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and

development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

73. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

74. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

75. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

75.1. Defect Workflow The diagram below shows the actual life cycle of a defect.

75.2. Defect States #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.

#2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.

#3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

gg)

Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.

hh) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of

the defect is changed to 'Duplicate' by the developer. ii) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

1. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

2. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

3. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

3.1. Defect Workflow The diagram below shows the actual life cycle of a defect.

3.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

a) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.

b) Duplicate: If the developer finds the defect as same

as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer. c) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'. d) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug". #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed". #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest". #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 3.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 4. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 5. Defect Management

#### 5.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

##### 5.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

##### 5.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

#### 5.2. Testing stages and defects Responsibilities

SL No. Testing Stage Environment Defect Owner Defect Retested By

Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team 4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager 5.3.Defect Triage Calls Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test. 5.3.1.Thumb rules of Triaging o All reported defects have been reviewed o All accepted defects have been prioritized o All accepted defects have severity attached o All rejected defects should have reasonable explanations for the testing team o Every defect has been assigned to an appropriate owner, individual or team o Root cause analysis for every accepted defect has been done 5.3.2.How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are: • The validity of the defect • Time sensitivity for resolution • The complexity involved in the resolution • Business impact This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced. 5.4.Defect Metrics Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be: • Defect distribution by cause • Defect distribution by feature/functional area • Defect distribution by Severity • Defect distribution by Priority • Defect distribution by type • Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user 5.5.Defect Reporting Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them. 5.5.1.Defect Trends by Release o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed. 5.5.2.Defects Density o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software. o Defect Density = Total Defect/Size of the Release 5.5.3.Defect Fix Rate per Release o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ . o Residual defect density = (total

number of defects found by a customer)/ (Total number of defects including customer found defects) x 100.

5.5.4. Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

5.5.5. Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.

Follow the steps below to create a fishbone diagram:

- (1) Write the problem at the head of the fish.
- (2) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
- (3) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
- (4) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement

Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit.

The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes.

Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

5.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects.

Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

5.6.1. Action items, Action Owners, Tracking, Checklist etc.

6. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was

resolved and recommendations for testing. • References 7. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted 3

A minor incident with low impact • A minor inconvenience to customers, workaround available In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs Blocker — Highest priority. Indicates that this issue takes precedence over all others. Critical — Indicates that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority. 8. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 9. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process. 10. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 11. Defect Management Life Cycle

When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it

goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

### 12. Defect Life Cycle in Detail

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

#### 12.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

#### 12.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.
- e) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- f) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.
- g) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.
- h) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".
- #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".
- #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".
- #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.
- #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.
- #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.
- #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

#### 12.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 13. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 14. Defect Management

#### 14.1. Defect Tracking

In software engineering, defect tracking is the process of

tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

14.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

14.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

14.2. Testing stages and defects Responsibilities SL No. Testing Stage Environment Defect Owner Defect Retested By Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team 4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager 14.3. Defect Triage Calls

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

14.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

14.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

14.4. Defect Metrics Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters.

Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

14.5. Defect Reporting Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

14.5.1. Defect Trends by Release o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

14.5.2. Defects Density o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

14.5.3. Defect Fix Rate per Release o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

14.5.4. Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

14.5.5. Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish. Follow the steps below to create a fishbone diagram:

- (5) Write the problem at the head of the fish.
- (6) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
- (7) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
- (8) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

14.6. Defect Prevention Processes & Improvements Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the

corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

14.6.1. Action items, Action Owners, Tracking, Checklist etc.

15. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was resolved and recommendations for testing.
- References

16. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples

- 1 A critical incident with very high impact
- A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss

2 A major incident with significant impact

- A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted

3 A minor incident with low impact

- A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

17. Defect Fix timeline

Severity Response Time Resolution Time

Severity	Response Time	Resolution Time
1	2	3
4	18.	Conclusion

This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

19. What is A Defect?

A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as

possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 20.

**Defect Management Life Cycle** When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project. 21.

**Defect Life Cycle in Detail** The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

**21.1. Defect Workflow** The diagram below shows the actual life cycle of a defect.

**21.2. Defect States #1) New:** This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle. **#2) Assigned:** In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer. **#3) Open:** Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason. **i) Rejected:** If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer. **j) Duplicate:** If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer. **k) Deferred:** If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'. **l) Not a Bug:** If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug". **#4) Fixed:** When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed". **#5) Pending Retest:** After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest". **#6) Retest:** At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. **#7) Reopen:** If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. **#8) Verified:** If the tester does not find any issue in the defect after being assigned to the

developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 21.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 22. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 23. Defect Management

#### 23.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

##### 23.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

##### 23.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

#### 23.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Retested By
1	Unit Testing (UT)	DEV	DEV Team	Dev Team
2	System Testing (ST)	ST	ST Team	ST Team
3	System Integration Testing (SIT)	SIT	SIT Team	SIT Team
4	User Acceptance Testing (UAT)	UAT	UAT Team	UAT Team

#### 23.3. Defect Triage

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

##### 23.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity

attached o All rejected defects should have reasonable explanations for the testing team o Every defect has been assigned to an appropriate owner, individual or team o Root cause analysis for every accepted defect has been done 23.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

23.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

23.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

23.5.1. Defect Trends by Release

- o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

23.5.2. Defects Density

- o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.
- o Defect Density = Total Defect/Size of the Release

23.5.3. Defect Fix Rate per Release

- o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .
- o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

23.5.4. Defect Aging

- o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

23.5.5. Defect Analysis through RCA

- o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.
- o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
- Follow the steps below to create a fishbone diagram:
- (9) Write the problem at the head of

the fish. (10) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N] (11) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N. (12) Extend the causes to secondary, tertiary, and more levels as applicable.

o Pareto charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement

Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit.

The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes.

Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

### 23.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects.

Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

#### 23.6.1. Action items, Action Owners, Tracking, Checklist etc.

24. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was resolved and recommendations for testing.
- References

25. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed.

Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder?

Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first?

Severity Description Examples

- 1 A critical incident with very high impact
- A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss

- 2 A major incident with significant impact
- A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted

- 3 A minor incident with low impact
- A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this

issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority. 26. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 27. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process. 28. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 29. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project. 30. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again. 30.1. Defect Workflow The diagram below shows the actual life cycle of a defect. 30.2. Defect States #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle. #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer. #3) Open: Here, the developer starts the

process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason. m)

Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer. n) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer. o) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'. p) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug". #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed". #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest". #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8)

Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter. 30.3. Guidelines for Implementing a Defect Life Cycle Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 31. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 32. Defect Management

32.1. Defect Tracking In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

32.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of

the work. Also, it enhances the efficient work delivered.

### 32.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

## 32.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Defect Retested By
1	Unit Testing (UT)	DEV	DEV Team	Dev Team
2	System Testing (ST)	ST	ST Team	ST Team
3	System Integration Testing (SIT)	SIT	SIT Team	SIT Team
4	User Acceptance Testing (UAT)	UAT	UAT Team	UAT Team
			Test Manager	

### 32.3. Defect Triage

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the “right” defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

#### 32.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

#### 32.3.2. How does Defect Triaging work?

The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

### 32.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

### 32.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

#### 32.5.1. Defect Trends by Release

- o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend

at which the defects are processed.

32.5.2. Defect Density o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

32.5.3. Defect Fix Rate per Release o Defect removal efficiency = (1 - (total defects caught by customer/ total number of defects)) x 100.

o Residual defect density = (total number of defects found by a customer)/ (Total number of defects including customer found defects) x 100.

32.5.4. Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

32.5.5. Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.

Follow the steps below to create a fishbone diagram:

- (13) Write the problem at the head of the fish.
- (14) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
- (15) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
- (16) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement. Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

32.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

32.6.1. Action items, Action Owners, Tracking, Checklist etc.

33. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or

Component where the Defect is introduced. • Project Activity occurs when the Defect is introduced. • Identification Method • Type of Defect • Projects and Products in which problems exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References

34. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples

1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss

2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted

3 A minor incident with low impact • A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

Blocker — Highest priority. Indicates that this issue takes precedence over all others.

Critical — Indicates that this issue is causing a problem and requires urgent attention.

Major — Indicates that this issue has a significant impact.

Minor — Indicates that this issue has a relatively minor impact.

Trivial — Lowest priority.

35. Defect Fix timeline

Severity Response Time Resolution Time

1	2	3
---	---	---

4 36. Conclusion

This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

37. What is A Defect?

A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

38. Defect Management Life Cycle

When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all

defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

### 39. Defect Life Cycle in Detail

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

#### 39.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

#### 39.2. Defect States

#1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.

#2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.

#3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

q) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.

r) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.

s) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.

t) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".

#4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".

#5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".

#6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.

#7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.

#8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.

#9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

#### 39.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that

particular defect can understand the reason of such a status of a defect very easily. • The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

#### 40. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

#### 41. Defect Management

##### 41.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

##### 41.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

##### 41.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

##### 41.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Defect Retested By
1	Unit Testing (UT)	DEV	DEV Team	Dev Team
2	System Testing (ST)	ST	ST	Team
3	System Integration Testing (SIT)	SIT	SIT Team	SIT Team
4	User Acceptance Testing (UAT)	UAT	UAT Team	UAT Team
				Test Manager

##### 41.3. Defect Triage

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the “right” defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

##### 41.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

##### 41.3.2. How does Defect Triaging work?

The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the

basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

#### 41.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

#### 41.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

##### 41.5.1. Defect Trends by Release

o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

##### 41.5.2. Defects Density

o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

##### 41.5.3. Defect Fix Rate per Release

o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

##### 41.5.4. Defect Aging

o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

##### 41.5.5. Defect Analysis through RCA

o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
- Follow the steps below to create a fishbone diagram:
  - (17) Write the problem at the head of the fish.
  - (18) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
  - (19) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
  - (20) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement.

Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

-

Identifying and prioritizing problems and tasks. • Helping people to organize their workloads more effectively. • Improving productivity. • Improving profitability. Pareto Analysis Steps 1. Identify and List Problems 2. Identify the Root Cause of Each Problem 3. Score Problems 4. Group Problems Together 5. Add up Scores for Each Group 6. Take Action 41.6. Defect Prevention Processes & Improvements Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks: #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products. #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system. #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects. #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis 41.6.1. Action items, Action Owners, Tracking, Checklist etc. 42. Defect Data • Name of the Person • Types of Testing • Problem Summary • Detailed Description of Defect. • Steps to Reproduce • Life Cycle Phase • Work product where Defect was introduced. • Severity and Priority • Subsystem or Component where the Defect is introduced. • Project Activity occurs when the Defect is introduced. • Identification Method • Type of Defect • Projects and Products in which problems exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References 43. Defect Priority and Severity Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted 3 A minor incident with low impact • A minor inconvenience to customers, workaround available In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs Blocker — Highest priority. Indicates that this issue takes precedence over all others. Critical — Indicates that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority. 44. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 5. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which

is collectively known as a Defect Management Process. 46. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 47.

Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project. 48. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again. 48.1. Defect Workflow The diagram below shows the actual life cycle of a defect. 48.2. Defect States #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle. #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer. #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason. u) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer. v) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer. w) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'. x) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug". #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed". #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to

48.2. Defect States #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle. #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer. #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason. u) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer. v) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer. w) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'. x) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug". #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed". #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to

retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest". #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

48.3. Guidelines for Implementing a Defect Life Cycle Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

49. Goals of Defect Management Process (DMP) Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

50. Defect Management 50.1. Defect Tracking In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

50.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

50.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

50.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Retested By
1	Unit Testing (UT)	DEV	DEV Team	Dev Team
2	System Testing (ST)	ST	ST Team	ST Team
3	System Integration Testing (SIT)	SIT	SIT Team	SIT Team
4	User Acceptance Testing (UAT)	UAT	UAT Team	UAT Team

50.3. Defect Triage Calls

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that

need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

50.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

50.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

50.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

50.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

50.5.1. Defect Trends by Release

- o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

50.5.2. Defects Density

- o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.
- o Defect Density = Total Defect/Size of the Release

50.5.3. Defect Fix Rate per Release

- o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .
- o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

50.5.4. Defect Aging

- o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

50.5.5. Defect Analysis through RCA

- o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.
- o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a

"fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms. Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish. Follow the steps below to create a fishbone diagram: (21)Write the problem at the head of the fish. (22)Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N] (23)Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N. (24)Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement

Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

#### 50.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

#### 50.6.1. Action items, Action Owners, Tracking, Checklist etc.

51. Defect Data
  - Name of the Person
  - Types of Testing
  - Problem Summary
  - Detailed Description of Defect.
  - Steps to Reproduce
  - Life Cycle Phase
  - Work product where Defect was introduced.
  - Severity and Priority
  - Subsystem or Component where the Defect is introduced.
  - Project Activity occurs when the Defect is introduced.
  - Identification Method
  - Type of Defect
  - Projects and Products in which problems exist
  - Current Owner
  - Current State of the Report
  - Work product where Defect occurred.
  - Impact on Project
  - Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
  - Dates when various defect lifecycle phases occur.
  - Description of how the defect was resolved and recommendations for testing.
  - References
52. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples

- 1 A critical incident with very high impact
- A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is

breached • Customer data loss 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted 3 A minor incident with low impact • A minor inconvenience to customers, workaround available In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs Blocker — Highest priority. Indicates that this issue takes precedence over all others. Critical — Indicates that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority. 53. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 54. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process. 55. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 56. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project. 57. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced

again.

57.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

57.2. Defect States

#1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.

#2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.

#3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

y) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.

z) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.

aa) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.

bb) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".

#4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".

#5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".

#6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.

#7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.

#8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.

#9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

57.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

58. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

59. Defect Management

59.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- ...

Created by • Created Date • Assigned to • Resolved Date • Resolved By • Status • Reason

59.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction. o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects. o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

59.1.2. Why is Defect Tracking Necessary? o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process. o Monitors the bugs that had already been resolved, thereby saving time. o To ensure that right defects are being worked on.

59.2. Testing stages and defects Responsibilities SL No. Testing Stage Environment Defect Owner Defect Retested By Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team

4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager

59.3. Defect Triage Calls Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

59.3.1. Thumb rules of Triaging o All reported defects have been reviewed o All accepted defects have been prioritized o All accepted defects have severity attached o All rejected defects should have reasonable explanations for the testing team o Every defect has been assigned to an appropriate owner, individual or team o Root cause analysis for every accepted defect has been done

59.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes.

Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

59.4. Defect Metrics Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters.

Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area

- Defect distribution by Severity
- Defect distribution by Priority

- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

59.5. Defect Reporting Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed.

Defect

reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

59.5.1. Defect Trends by Release o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

59.5.2. Defects Density o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

59.5.3. Defect Fix Rate per Release o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

59.5.4. Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

59.5.5. Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
- Follow the steps below to create a fishbone diagram:
  - (25) Write the problem at the head of the fish.
  - (26) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
  - (27) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
  - (28) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement

Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit.

The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes.

Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

59.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects.

Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and

Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.

#4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

59.6.1. Action items, Action Owners, Tracking, Checklist etc.

60. Defect Data • Name of the Person • Types of Testing • Problem Summary • Detailed Description of Defect. • Steps to Reproduce • Life Cycle Phase • Work product where Defect was introduced. • Severity and Priority • Subsystem or Component where the Defect is introduced. • Project Activity occurs when the Defect is introduced. • Identification Method • Type of Defect • Projects and Products in which problems exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References

61. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first?

Severity Description Examples

- 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss

- 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted

- 3 A minor incident with low impact • A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

62. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 63. Conclusion

This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

64. What is A Defect?

A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

65. Defect Management Life Cycle

When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software

Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

#### 66. Defect Life Cycle in Detail

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

##### 66.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

##### 66.2. Defect States #1)

New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.

#2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.

#3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

cc) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.

dd) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.

ee) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.

ff) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".

#4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".

#5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".

#6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.

#7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.

#8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.

#9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

##### 66.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly

understands the different states of a defect (discussed above). • Defect Life Cycle should be properly documented to avoid any confusion in the future. • Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results. • Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily. • The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

67. Goals of Defect Management Process (DMP) Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

68. Defect Management

68.1. Defect Tracking In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

68.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

68.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

68.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Defect Retested By
1	Unit Testing (UT)	DEV	DEV Team	Dev Team
2	System Testing (ST)	ST	ST Team	ST Team
3	System Integration Testing (SIT)	SIT	SIT Team	SIT Team
4	User Acceptance Testing (UAT)	UAT	UAT Team	UAT Team

68.3. Defect Triage

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

68.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

68.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity

corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

#### 68.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

#### 68.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

##### 68.5.1. Defect Trends by Release

o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

##### 68.5.2. Defects Density

o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

##### 68.5.3. Defect Fix Rate per Release

o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

##### 68.5.4. Defect Aging

o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

##### 68.5.5. Defect Analysis through RCA

o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
- Follow the steps below to create a fishbone diagram:
  - (29) Write the problem at the head of the fish.
  - (30) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
  - (31) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
  - (32) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement

Pareto Analysis is a simple decision-making technique for

assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

#### 6.8.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

#### 6.8.6.1. Action items, Action Owners, Tracking, Checklist etc.

6.9. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was resolved and recommendations for testing.
- References

#### 7.0. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first?

Severity Description Examples

- 1 A critical incident with very high impact
- A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss

- 2 A major incident with significant impact
- A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted

- 3 A minor incident with low impact
- A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

#### 7.1. Defect Fix timeline

Severity Response Time Resolution Time

1	2	3
4	72.	Conclusion

This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best

form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

73. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

74. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse.

In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

75. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

75.1. Defect Workflow The diagram below shows the actual life cycle of a defect.

75.2. Defect States #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle. #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer. #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

gg)  
Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.  
hh) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.  
ii) Deferred: If the developer feels that

the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

1. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

2. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

3. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

3.1. Defect Workflow The diagram below shows the actual life cycle of a defect.

3.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

a) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.

b) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of

the defect is changed to 'Duplicate' by the developer. c) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'. d) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug". #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed". #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest". #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 3.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 4. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 5. Defect Management

#### 5.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

##### 5.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

##### 5.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

#### 5.2. Testing stages and defects Responsibilities

SL No. Testing Stage Environment Defect Owner Defect Retested By Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team

ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team 4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager 5.3.Defect Triage Calls Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test. 5.3.1.Thumb rules of Triaging o All reported defects have been reviewed o All accepted defects have been prioritized o All accepted defects have severity attached o All rejected defects should have reasonable explanations for the testing team o Every defect has been assigned to an appropriate owner, individual or team o Root cause analysis for every accepted defect has been done 5.3.2.How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are: • The validity of the defect • Time sensitivity for resolution • The complexity involved in the resolution • Business impact This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced. 5.4.Defect Metrics Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be: • Defect distribution by cause • Defect distribution by feature/functional area • Defect distribution by Severity • Defect distribution by Priority • Defect distribution by type • Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user 5.5.Defect Reporting Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them. 5.5.1.Defect Trends by Release o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed. 5.5.2.Defects Density o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software. o Defect Density = Total Defect/Size of the Release 5.5.3.Defect Fix Rate per Release o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ . o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found})$

defects) x 100.

5.5.4. Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

5.5.5. Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.

Follow the steps below to create a fishbone diagram:

- (1) Write the problem at the head of the fish.
- (2) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
- (3) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
- (4) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement

Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit.

The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes.

Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

5.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects.

Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

5.6.1. Action items, Action Owners, Tracking, Checklist etc.

6. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was resolved and recommendations for testing.
- References

7. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted 3 A minor incident with low impact • A minor inconvenience to customers, workaround available In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs Blocker — Highest priority. Indicates that this issue takes precedence over all others. Critical — Indicates that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority.

8. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 9. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

10. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

11. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle.

Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

### 12. Defect Life Cycle in Detail

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

#### 12.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

#### 12.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.
- e) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- f) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.
- g) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.
- h) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".
- #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".
- #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".
- #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.
- #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.
- #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.
- #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 12.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 13. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 14. Defect Management

#### 14.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by

inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier.

Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

14.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

14.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

14.2. Testing stages and defects Responsibilities SL No. Testing Stage Environment Defect Owner Defect Retested By Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team 4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager

14.3. Defect Triage Calls Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

14.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

14.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

14.4. Defect Metrics Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect

distribution by feature/functional area • Defect distribution by Severity • Defect distribution by Priority • Defect distribution by type • Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

14.5. Defect Reporting Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

14.5.1. Defect Trends by Release o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

14.5.2. Defects Density o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

14.5.3. Defect Fix Rate per Release o Defect removal efficiency = (1 - (total defects caught by customer/ total number of defects)) x 100.

o Residual defect density = (total number of defects found by a customer)/ (Total number of defects including customer found defects) x 100.

14.5.4. Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

14.5.5. Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish. Follow the steps below to create a fishbone diagram:

- (5) Write the problem at the head of the fish.
- (6) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
- (7) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
- (8) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement

Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit.

The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes.

Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

14.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects.

Defect Prevention is

a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

14.6.1. Action items, Action Owners, Tracking, Checklist etc.

15. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was resolved and recommendations for testing.
- References

16. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first?

Severity Description Examples

- 1 A critical incident with very high impact
- A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss

2 A major incident with significant impact

- A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted

3 A minor incident with low impact

- A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

17. Defect Fix timeline

Severity Response Time Resolution Time

1	2	3
4	18. Conclusion	

This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

19. What is A Defect?

A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand

the defect life cycle before moving to the workflow and different states of the defect. 20.

**Defect Management Life Cycle** When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project. 21.

**Defect Life Cycle in Detail** The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

**21.1. Defect Workflow** The diagram below shows the actual life cycle of a defect. 21.2. **Defect States #1) New:** This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle. **#2) Assigned:** In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer. **#3) Open:** Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

i) **Rejected:** If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.

j) **Duplicate:** If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.

k) **Deferred:** If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.

l) **Not a Bug:** If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".

#4) **Fixed:** When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".

#5) **Pending Retest:** After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".

#6) **Retest:** At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.

#7) **Reopen:** If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.

#8) **Verified:** If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status

of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 21.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 22. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 23. Defect Management

#### 23.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

##### 23.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

##### 23.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

#### 23.2. Testing stages and defects Responsibilities

SL No. Testing Stage Environment Defect Owner Defect Retested By Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team 4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager

### 23.3. Defect Triage

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

#### 23.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team o

Every defect has been assigned to an appropriate owner, individual or team o Root cause analysis for every accepted defect has been done 23.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

### 23.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

### 23.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

#### 23.5.1. Defect Trends by Release

o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

#### 23.5.2. Defects Density

o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

#### 23.5.3. Defect Fix Rate per Release

o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

#### 23.5.4. Defect Aging

o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

#### 23.5.5. Defect Analysis through RCA

o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
- Follow the steps below to create a fishbone diagram:
  - (9) Write the problem at the head of the fish.
  - (10) Identify the category of causes and write at end of each bone [cause category 1,

cause category 2 ..... cause category N] (11)Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N. (12)Extend the causes to secondary, tertiary, and more levels as applicable. o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

### 23.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

#### 23.6.1. Action items, Action Owners, Tracking, Checklist etc.

24. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced
- Severity and Priority
- Subsystem or Component where the Defect is introduced
- Project Activity occurs when the Defect is introduced
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect
- Dates when various defect lifecycle phases occur
- Description of how the defect was resolved and recommendations for testing
- References

25. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first?

Severity Description Examples

- 1 A critical incident with very high impact
- A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss

- 2 A major incident with significant impact
- A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted

- 3 A minor incident with low impact
- A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.

Trivial — Lowest priority. 26. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 27. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process. 28. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 29. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project. 30. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again. 30.1. Defect Workflow The diagram below shows the actual life cycle of a defect. 30.2. Defect States #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle. #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer. #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that

the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason. m)

Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer. n) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer. o) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'. p) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug". #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".

#5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest". #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8)

Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

30.3. Guidelines for Implementing a Defect Life Cycle Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

31. Goals of Defect Management Process (DMP) Given below are the various goals of this process:

- Prevent the Defect

- Early Detection

- Minimize the impact

- Resolution of the Defect

- Process improvement

32. Defect Management 32.1. Defect Tracking In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier.

Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

32.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

32.1.2. Why is Defect Tracking

Necessary? o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process. o Monitors the bugs that had already been resolved, thereby saving time. o To ensure that right defects are being worked on.

### 32.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Defect Retested By
1	Unit Testing (UT)	DEV	DEV Team	Dev Team
2	System Testing (ST)	ST	ST	ST
3	System Integration Testing (SIT)	SIT	SIT	Team SIT
4	User Acceptance Testing (UAT)	UAT	UAT Team	UAT Team
				Test Manager

### 32.3. Defect Triage

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the “right” defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

#### 32.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

#### 32.3.2. How does Defect Triaging work?

The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

### 32.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

### 32.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

#### 32.5.1. Defect Trends by Release

- o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

#### 32.5.2. Defects Density

- o Defect density can be defined as

the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

- o Defect Density = Total Defect/Size of the Release
- 32.5.3. Defect Fix Rate per Release
- o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .
- o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

32.5.4. Defect Aging

- o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

32.5.5. Defect Analysis through RCA

- o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.
- o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
- Follow the steps below to create a fishbone diagram:
- (13) Write the problem at the head of the fish.
- (14) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
- (15) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
- (16) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement.

Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit.

The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes.

Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

32.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects.

Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

32.6.1. Action items, Action Owners, Tracking, Checklist etc.

33. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is

introduced. • Identification Method • Type of Defect • Projects and Products in which problems exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References

34. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples

1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss

2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted

3 A minor incident with low impact • A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

Blocker — Highest priority. Indicates that this issue takes precedence over all others.

Critical — Indicates that this issue is causing a problem and requires urgent attention.

Major — Indicates that this issue has a significant impact.

Minor — Indicates that this issue has a relatively minor impact.

Trivial — Lowest priority.

35. Defect Fix timeline

Severity Response Time Resolution Time 1 2 3 4

36. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

37. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

38. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected,

postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

### 39. Defect Life Cycle in Detail

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

#### 39.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

#### 39.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.
- q) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- r) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.
- s) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.
- t) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".
- #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".
- #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".
- #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.
- #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.
- #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.
- #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

#### 39.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The

defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

#### 40. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

#### 41. Defect Management

##### 41.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

##### 41.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

##### 41.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

#### 41.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Defect Retested By
1	Unit Testing (UT)	DEV	DEV Team	Dev Team
2	System Testing (ST)	ST	ST Team	ST Team
3	System Integration Testing (SIT)	SIT	SIT Team	SIT Team
4	User Acceptance Testing (UAT)	UAT	UAT Team	UAT Team

#### 41.3. Defect Triage Calls

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

##### 41.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

##### 41.3.2. How does Defect Triaging work?

The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every

single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

#### 41.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

#### 41.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

##### 41.5.1. Defect Trends by Release

- o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

##### 41.5.2. Defects Density

- o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.
- o Defect Density = Total Defect/Size of the Release

##### 41.5.3. Defect Fix Rate per Release

- o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .
- o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

##### 41.5.4. Defect Aging

- o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

##### 41.5.5. Defect Analysis through RCA

- o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.
- o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.
- Steps to create a Fishbone Diagram:
  - Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
  - Follow the steps below to create a fishbone diagram:
    - (17) Write the problem at the head of the fish.
    - (18) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
    - (19) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
    - (20) Extend the causes to secondary, tertiary, and more levels as applicable.
- o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement. Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads

more effectively. • Improving productivity. • Improving profitability. Pareto Analysis Steps 1. Identify and List Problems 2. Identify the Root Cause of Each Problem 3. Score Problems 4. Group Problems Together 5. Add up Scores for Each Group 6. Take Action 41.6. Defect Prevention Processes & Improvements Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks: #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products. #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system. #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects. #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis 41.6.1. Action items, Action Owners, Tracking, Checklist etc. 42. Defect Data • Name of the Person • Types of Testing • Problem Summary • Detailed Description of Defect. • Steps to Reproduce • Life Cycle Phase • Work product where Defect was introduced. • Severity and Priority • Subsystem or Component where the Defect is introduced. • Project Activity occurs when the Defect is introduced. • Identification Method • Type of Defect • Projects and Products in which problems exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References 43. Defect Priority and Severity Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted 3 A minor incident with low impact • A minor inconvenience to customers, workaround available In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs Blocker — Highest priority. Indicates that this issue takes precedence over all others. Critical — Indicates that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority. 44. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 45. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process. 46. What is A Defect? A Defect, in

simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 47.

**Defect Management Life Cycle** When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project. 48. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again. 48.1. Defect Workflow The diagram below shows the actual life cycle of a defect. 48.2. Defect States #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle. #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer. #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason. u) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer. v) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer. w) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'. x) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug". #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed". #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of

the defect remains in "Pending Retest". #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 48.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 49. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 50. Defect Management

#### 50.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

##### 50.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

##### 50.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

#### 50.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Retested By
1	Unit Testing (UT)	DEV	DEV Team	Dev Team
2	System Testing (ST)	ST	ST Team	ST Team
3	System Integration Testing (SIT)	SIT	SIT Team	SIT Team
4	User Acceptance Testing (UAT)	UAT	UAT Team	UAT Team

#### 50.3. Defect Triage Calls

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in

preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

50.3.1. Thumb rules of Triaging o All reported defects have been reviewed o All accepted defects have been prioritized o All accepted defects have severity attached o All rejected defects should have reasonable explanations for the testing team o Every defect has been assigned to an appropriate owner, individual or team o Root cause analysis for every accepted defect has been done

50.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

50.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

50.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

50.5.1. Defect Trends by Release

o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

50.5.2. Defects Density

o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

50.5.3. Defect Fix Rate per Release

o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

50.5.4. Defect Aging

o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

50.5.5. Defect Analysis through RCA

o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in

sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms. Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish. Follow the steps below to create a fishbone diagram: (21)Write the problem at the head of the fish. (22)Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N] (23)Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N. (24)Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement

Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit.

The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes.

Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

#### 50.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects.

Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

#### 50.6.1. Action items, Action Owners, Tracking, Checklist etc.

51. Defect Data
  - Name of the Person
  - Types of Testing
  - Problem Summary
  - Detailed Description of Defect
  - Steps to Reproduce
  - Life Cycle Phase
  - Work product where Defect was introduced
  - Severity and Priority
  - Subsystem or Component where the Defect is introduced
  - Project Activity occurs when the Defect is introduced
  - Identification Method
  - Type of Defect
  - Projects and Products in which problems exist
  - Current Owner
  - Current State of the Report
  - Work product where Defect occurred
  - Impact on Project
  - Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect
  - Dates when various defect lifecycle phases occur
  - Description of how the defect was resolved and recommendations for testing
  - References
52. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first?

Severity Description Examples

- 1 A critical incident with very high impact
  - A customer-facing service, like Jira Cloud, is down for all customers
  - Confidentiality or privacy is breached
  - Customer data loss
- 2 A major incident with significant impact
  - A customer-facing

service is unavailable for a subset of customers • Core functionality is significantly impacted 3 A minor incident with low impact • A minor inconvenience to customers, workaround available In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs Blocker — Highest priority. Indicates that this issue takes precedence over all others. Critical — Indicates that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority. 53. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 54. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process. 55. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 56. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project. 57. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again. 57.1. Defect Workflow The diagram below shows the actual life cycle of a defect. 57.2.

Defect States #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle. #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer. #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason. y) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer. z) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer. aa) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'. bb) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug". #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed". #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest". #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

57.3. Guidelines for Implementing a Defect Life Cycle Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

58. Goals of Defect Management Process (DMP) Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

59. Defect Management

59.1. Defect Tracking In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

59.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction. o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects. o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

59.1.2. Why is Defect Tracking Necessary? o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process. o Monitors the bugs that had already been resolved, thereby saving time. o To ensure that right defects are being worked on.

59.2. Testing stages and defects Responsibilities SL No. Testing Stage Environment Defect Owner Defect Retested By

Defect Tracking	By	1	Unit Testing (UT)	DEV	DEV Team	Dev Team	2	System Testing (ST)	ST	ST			
Team	ST	Team	ST	Team	3	System Integration Testing (SIT)	SIT	SIT	Team	SIT	Team	SIT	Team
4	User Acceptance Testing (UAT)	UAT	UAT Team	UAT Team	Test Manager								

59.3. Defect Triage Calls

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

59.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

59.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

59.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

59.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management

board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

59.5.1. Defect Trends by Release o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

59.5.2. Defects Density o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

59.5.3. Defect Fix Rate per Release o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

59.5.4. Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

59.5.5. Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
- Follow the steps below to create a fishbone diagram:
  - (25) Write the problem at the head of the fish.
  - (26) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
  - (27) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
  - (28) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit.

The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

59.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can

be used to support analyzing defects. #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

59.6.1. Action items, Action Owners, Tracking, Checklist etc.

60. Defect Data • Name of the Person • Types of Testing • Problem Summary • Detailed Description of Defect. • Steps to Reproduce • Life Cycle Phase • Work product where Defect was introduced. • Severity and Priority • Subsystem or Component where the Defect is introduced. • Project Activity occurs when the Defect is introduced. • Identification Method • Type of Defect • Projects and Products in which problems exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References

61. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples

1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss

2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted

3 A minor incident with low impact • A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

Blocker — Highest priority. Indicates that this issue takes precedence over all others. Critical — Indicates that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority.

62. Defect Fix timeline

Severity Response Time Resolution Time 1 2 3

4 63. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process

Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

64. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

65. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a

flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

### 66. Defect Life Cycle in Detail

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

#### 66.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

#### 66.2. Defect States #1) New:

This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.

#### #2) Assigned:

In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.

#### #3) Open:

Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

cc)

#### Rejected:

If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.

dd) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.

ee) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.

ff) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".

#### #4) Fixed:

When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".

#### #5) Pending Retest:

After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".

#### #6) Retest:

At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.

#### #7) Reopen:

If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.

#### #8) Verified:

If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.

#### #9) Closed:

When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 66.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be

properly documented to avoid any confusion in the future. • Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results. • Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily. • The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

67. Goals of Defect Management Process (DMP) Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

68. Defect Management

68.1. Defect Tracking In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

68.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

68.1.2. Why is Defect Tracking Necessary?

- A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- Monitors the bugs that had already been resolved, thereby saving time.
- To ensure that right defects are being worked on.

68.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Defect Retested By
1	Unit Testing (UT)	DEV	DEV Team	Dev Team
2	System Testing (ST)	ST	ST Team	ST Team
3	System Integration Testing (SIT)	SIT	SIT Team	SIT Team
4	User Acceptance Testing (UAT)	UAT	UAT Team	UAT Team

68.3. Defect Triage Calls

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the “right” defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

68.3.1. Thumb rules of Triaging

- All reported defects have been reviewed
- All accepted defects have been prioritized
- All accepted defects have severity attached
- All rejected defects should have reasonable explanations for the testing team
- Every defect has been assigned to an appropriate owner, individual or team
- Root cause analysis for every accepted defect has been done

68.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected

at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

#### 68.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

#### 68.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

##### 68.5.1. Defect Trends by Release

o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

##### 68.5.2. Defects Density

o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

$$\text{o Defect Density} = \frac{\text{Total Defect}}{\text{Size of the Release}}$$

##### 68.5.3. Defect Fix Rate per Release

o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

##### 68.5.4. Residual defect density

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

##### 68.5.4. Defect Aging

o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

##### 68.5.5. Defect Analysis through RCA

o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
- Follow the steps below to create a fishbone diagram:
  - (29) Write the problem at the head of the fish.
  - (30) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
  - (31) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
  - (32) Extend the causes to secondary, tertiary, and more levels as applicable.
- o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement.

Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to

focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

68.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

68.6.1. Action items, Action Owners, Tracking, Checklist etc.

69. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was resolved and recommendations for testing.
- References

70. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first?

Severity Description Examples

- 1 A critical incident with very high impact
- A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss
- 2 A major incident with significant impact
- A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted
- 3 A minor incident with low impact
- A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

71. Defect Fix timeline

Severity Response Time Resolution Time

1	2	3
4	72. Conclusion	This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process

Defect management process is mainly used to improve the

efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

73. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

74. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

75. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

75.1. Defect Workflow The diagram below shows the actual life cycle of a defect.

75.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

gg)  
Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.  
hh) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.  
ii) Deferred: If the developer feels that

the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.

**Defect Management Process** Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

1. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

2. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

3. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

3.1. Defect Workflow The diagram below shows the actual life cycle of a defect.

3.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

a) Rejected: If the defect is not considered a genuine defect by the developer, then it is

marked as "Rejected" by the developer. b) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer. c) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'. d) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug". #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed". #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest". #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 3.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 4. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 5. Defect Management

#### 5.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

##### 5.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

##### 5.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

#### 5.2. Testing stages and defects

Responsibilities SL No. Testing Stage Environment Defect Owner Defect Retested By Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team 4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager 5.3.Defect Triage Calls Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test. 5.3.1.Thumb rules of Triaging o All reported defects have been reviewed o All accepted defects have been prioritized o All accepted defects have severity attached o All rejected defects should have reasonable explanations for the testing team o Every defect has been assigned to an appropriate owner, individual or team o Root cause analysis for every accepted defect has been done 5.3.2.How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are: • The validity of the defect • Time sensitivity for resolution • The complexity involved in the resolution • Business impact This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced. 5.4.Defect Metrics Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be: • Defect distribution by cause • Defect distribution by feature/functional area • Defect distribution by Severity • Defect distribution by Priority • Defect distribution by type • Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user 5.5.Defect Reporting Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them. 5.5.1.Defect Trends by Release o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed. 5.5.2.Defects Density o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software. o Defect Density = Total Defect/Size of the Release 5.5.3.Defect Fix Rate per Release o Defect removal efficiency = (1 - (total defects

caught by customer/ total number of defects)) x 100. o Residual defect density = (total number of defects found by a customer)/ (Total number of defects including customer found defects) x 100.

5.5.4. Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

5.5.5. Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.

Follow the steps below to create a fishbone diagram:

- (1) Write the problem at the head of the fish.
- (2) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
- (3) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
- (4) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement

Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit.

The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes.

Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

5.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects.

Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

5.6.1. Action items, Action Owners, Tracking, Checklist etc.

6. Defect Data
  - Name of the Person
  - Types of Testing
  - Problem Summary
  - Detailed Description of Defect.
  - Steps to Reproduce
  - Life Cycle Phase
  - Work product where Defect was introduced.
  - Severity and Priority
  - Subsystem or Component where the Defect is introduced.
  - Project Activity occurs when the Defect is introduced.
  - Identification Method
  - Type of Defect
  - Projects and Products in which problems exist
  - Current Owner
  - Current State of the Report
  - Work product where Defect occurred.
  - Impact on Project
  - Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.

- Dates when various defect lifecycle phases occur.
  - Description of how the defect was resolved and recommendations for testing.
  - References
7. Defect Priority and Severity
- Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples
- 1 A critical incident with very high impact
  - A customer-facing service, like Jira Cloud, is down for all customers
  - Confidentiality or privacy is breached
  - Customer data loss
  - 2 A major incident with significant impact
  - A customer-facing service is unavailable for a subset of customers
  - Core functionality is significantly impacted
  - 3 A minor incident with low impact
  - A minor inconvenience to customers, workaround available
- In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs
- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
  - Critical — Indicates that this issue is causing a problem and requires urgent attention.
  - Major — Indicates that this issue has a significant impact.
  - Minor — Indicates that this issue has a relatively minor impact.
  - Trivial — Lowest priority.
8. Defect Fix timeline
- | Severity | Time Resolution |
|----------|-----------------|
| 1        | 1               |
| 2        | 2               |
| 3        | 3               |
9. Conclusion
- This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process
- Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.
10. What is A Defect?
- A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.
11. Defect Management Life Cycle
- When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and

standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

### 12. Defect Life Cycle in Detail

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

#### 12.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

#### 12.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.
- e) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- f) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.
- g) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.
- h) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".
- #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".
- #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".
- #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.
- #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.
- #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.
- #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 12.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 13. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 14. Defect

**Management 14.1. Defect Tracking** In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

**14.1.1. Objectives of Defect Tracking** A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

**14.1.2. Why is Defect Tracking Necessary?** o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.

o Monitors the bugs that had already been resolved, thereby saving time.

o To ensure that right defects are being worked on.

**14.2. Testing stages and defects Responsibilities** SL No. Testing Stage Environment Defect Owner Defect Retested By

Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST

Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team

4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager

**14.3. Defect Triage Calls** Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

**14.3.1. Thumb rules of Triaging** o All reported defects have been reviewed

o All accepted defects have been prioritized

o All accepted defects have severity attached

o All rejected defects should have reasonable explanations for the testing team

o Every defect has been assigned to an appropriate owner, individual or team

o Root cause analysis for every accepted defect has been done

**14.3.2. How does Defect Triaging work?** The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

**14.4. Defect Metrics** Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs

have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

14.5. Defect Reporting Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

14.5.1. Defect Trends by Release o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

14.5.2. Defects Density o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

14.5.3. Defect Fix Rate per Release o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

14.5.4. Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

14.5.5. Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish. Follow the steps below to create a fishbone diagram:

- (5) Write the problem at the head of the fish.
- (6) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
- (7) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
- (8) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

14.6. Defect Prevention Processes & Improvements Defect prevention is a framework and ongoing process

of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

14.6.1. Action items, Action Owners, Tracking, Checklist etc.

15. Defect Data • Name of the Person • Types of Testing • Problem Summary • Detailed Description of Defect. • Steps to Reproduce • Life Cycle Phase • Work product where Defect was introduced. • Severity and Priority • Subsystem or Component where the Defect is introduced. • Project Activity occurs when the Defect is introduced. • Identification Method • Type of Defect • Projects and Products in which problems exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References

16. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first?

Severity Description Examples

- 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss
- 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted
- 3 A minor incident with low impact • A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

17. Defect Fix timeline

Severity	Response Time	Resolution Time
1	2	3

18. Conclusion

This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

19. What is A Defect?

A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the

responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 20.

**Defect Management Life Cycle** When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project. 21. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again. 21.1. Defect Workflow The diagram below shows the actual life cycle of a defect. 21.2. Defect States #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle. #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer. #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason. i) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer. j) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer. k) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'. l) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug". #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed". #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest". #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8)

Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

21.3. Guidelines for Implementing a Defect Life Cycle Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

22. Goals of Defect Management Process (DMP) Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

23. Defect Management

23.1. Defect Tracking In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

23.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

23.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

23.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Defect Retested By
1	Unit Testing (UT)	DEV	DEV Team	Dev Team
2	System Testing (ST)	ST	ST Team	ST Team
3	System Integration Testing (SIT)	SIT	SIT Team	SIT Team
4	User Acceptance Testing (UAT)	UAT	UAT Team	UAT Team
			Test Manager	

23.3. Defect Triage Calls

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

23.3.1. Thumb rules of Triaging

- o All reported defects have been

reviewed o All accepted defects have been prioritized o All accepted defects have severity attached o All rejected defects should have reasonable explanations for the testing team o Every defect has been assigned to an appropriate owner, individual or team o Root cause analysis for every accepted defect has been done 23.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

23.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

23.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

23.5.1. Defect Trends by Release

o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

23.5.2. Defects Density

o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

23.5.3. Defect Fix Rate per Release

o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

23.5.4. Defect Aging

o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

23.5.5. Defect Analysis through RCA

o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram

A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the

fish. Follow the steps below to create a fishbone diagram: (9) Write the problem at the head of the fish. (10) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N] (11) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N. (12) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

### 23.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

#### 23.6.1. Action items, Action Owners, Tracking, Checklist etc.

24. Defect Data
  - Name of the Person
  - Types of Testing
  - Problem Summary
  - Detailed Description of Defect.
  - Steps to Reproduce
  - Life Cycle Phase
  - Work product where Defect was introduced.
  - Severity and Priority
  - Subsystem or Component where the Defect is introduced.
  - Project Activity occurs when the Defect is introduced.
  - Identification Method
  - Type of Defect
  - Projects and Products in which problems exist
  - Current Owner
  - Current State of the Report
  - Work product where Defect occurred.
  - Impact on Project
  - Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
  - Dates when various defect lifecycle phases occur.
  - Description of how the defect was resolved and recommendations for testing.
  - References
25. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples

- 1 A critical incident with very high impact
- A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss
- 2 A major incident with significant impact
- A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted
- 3 A minor incident with low impact
- A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates

that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority.

26. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 27. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

28. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

29. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

30. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

30.1. Defect Workflow The diagram below shows the actual life cycle of a defect.

30.2. Defect States #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle. #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA

or test lead of the testing team to a developer. #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason. m) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer. n) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer. o) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'. p) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug". #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed". #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest". #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 30.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 31. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 32. Defect Management

#### 32.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

#### 32.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to

avoid further defects. o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

### 32.1.2. Why is Defect Tracking Necessary?

o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.

o Monitors the bugs that had already been resolved, thereby saving time.

o To ensure that right defects are being worked on.

### 32.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Defect Retested By
1	Unit Testing (UT)	DEV	DEV Team	Dev Team
2	System Testing (ST)	ST	ST Team	ST Team
3	System Integration Testing (SIT)	SIT	SIT Team	SIT Team
4	User Acceptance Testing (UAT)	UAT	UAT Team	UAT Team

### 32.3. Defect Triage Calls

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

#### 32.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

#### 32.3.2. How does Defect Triaging work?

The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

### 32.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

### 32.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

#### 32.5.1. Defect Trends by Release

- o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and

closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

32.5.2. Defect Density o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

32.5.3. Defect Fix Rate per Release o Defect removal efficiency = (1 - (total defects caught by customer/ total number of defects)) x 100.

o Residual defect density = (total number of defects found by a customer)/ (Total number of defects including customer found defects) x 100.

32.5.4. Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

32.5.5. Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
- Follow the steps below to create a fishbone diagram:

- (13) Write the problem at the head of the fish.
- (14) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
- (15) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
- (16) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement

Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit.

The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes.

Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

32.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

32.6.1. Action items, Action Owners, Tracking, Checklist etc.

33. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect
- Steps to Reproduce
- Life Cycle

Phase • Work product where Defect was introduced. • Severity and Priority • Subsystem or Component where the Defect is introduced. • Project Activity occurs when the Defect is introduced. • Identification Method • Type of Defect • Projects and Products in which problems exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References

34. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples

- 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss
- 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted
- 3 A minor incident with low impact • A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

35. Defect Fix timeline

Severity Response Time Resolution Time 1 2 3

4 36. Conclusion

This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process

Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

37. What is A Defect?

A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

38. Defect Management Life Cycle

When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the

status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

### 39. Defect Life Cycle in Detail

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

#### 39.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

#### 39.2. Defect States

#1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.

#2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.

#3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

q) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.

r) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.

s) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.

t) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".

#4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".

#5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".

#6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.

#7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.

#8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.

#9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

#### 39.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the

status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily. • The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

40. Goals of Defect Management Process (DMP) Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

41. Defect Management

41.1. Defect Tracking In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

41.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

41.1.2. Why is Defect Tracking Necessary?

- A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- Monitors the bugs that had already been resolved, thereby saving time.
- To ensure that right defects are being worked on.

41.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Defect Retested By
1	Unit Testing (UT)	DEV	DEV Team	Dev Team
2	System Testing (ST)	ST Team	ST Team	ST Team
3	System Integration Testing (SIT)	SIT Team	SIT Team	SIT Team
4	User Acceptance Testing (UAT)	UAT Team	UAT Team	Test Manager

41.3. Defect Triage

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

41.3.1. Thumb rules of Triaging

- All reported defects have been reviewed
- All accepted defects have been prioritized
- All accepted defects have severity attached
- All rejected defects should have reasonable explanations for the testing team
- Every defect has been assigned to an appropriate owner, individual or team
- Root cause analysis for every accepted defect has been done

41.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all

relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

#### 41.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

#### 41.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

##### 41.5.1. Defect Trends by Release

o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

##### 41.5.2. Defects Density

o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

##### 41.5.3. Defect Fix Rate per Release

o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

##### 41.5.4. Defect Aging

o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

##### 41.5.5. Defect Analysis through RCA

o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
- Follow the steps below to create a fishbone diagram:
  - (17) Write the problem at the head of the fish.
  - (18) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
  - (19) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
  - (20) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement.

Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit.

The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem

areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

#### 41.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

#### 41.6.1. Action items, Action Owners, Tracking, Checklist etc.

42. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was resolved and recommendations for testing.
- References

43. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first?

Severity Description Examples

- 1 A critical incident with very high impact
- A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss

- 2 A major incident with significant impact
- A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted

- 3 A minor incident with low impact
- A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

44. Defect Fix timeline

Severity Response Time Resolution Time

- 1
- 2
- 3

45. Conclusion

This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The

organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process. 46. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 47.

Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project. 48. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again. 48.1. Defect Workflow The diagram below shows the actual life cycle of a defect. 48.2. Defect States #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle. #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer. #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason. u) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer. v) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer. w) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'. x) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug". #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".

#5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest". #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

48.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

49. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

50. Defect Management

50.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

50.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

50.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

50.2. Testing stages and defects Responsibilities

SL No. Testing Stage Environment Defect Owner Defect Retested By

Defect Tracking By

- 1 Unit Testing (UT) DEV DEV Team Dev Team
- 2 System Testing (ST) ST ST Team
- 3 ST Team ST Team
- 4 System Integration Testing (SIT) SIT SIT Team SIT Team
- 5 SIT Team SIT Team
- 6 User Acceptance Testing (UAT) UAT UAT Team UAT Team
- 7 Test Manager

50.3. Defect Triage Calls

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help

focus the team on the “right” defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

50.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team

Every defect has been assigned to an appropriate owner, individual or team

- o Root cause analysis for every accepted defect has been done

50.3.2. How does Defect Triaging work?

The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

50.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

50.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

50.5.1. Defect Trends by Release

- o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

50.5.2. Defects Density

- o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.
- o  $\text{Defect Density} = \frac{\text{Total Defect}}{\text{Size of the Release}}$

50.5.3. Defect Fix Rate per Release

- o  $\text{Defect removal efficiency} = (1 - (\frac{\text{total defects caught by customer}}{\text{total number of defects}})) \times 100$ .
- o  $\text{Residual defect density} = \frac{(\text{total number of defects found by a customer})}{(\text{Total number of defects including customer found defects})} \times 100$ .

50.5.4. Defect Aging

- o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

50.5.5. Defect Analysis through RCA

- o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the

organization.

- o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms. Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish. Follow the steps below to create a fishbone diagram: (21)Write the problem at the head of the fish. (22)Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N] (23)Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N. (24)Extend the causes to secondary, tertiary, and more levels as applicable.
- o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

#### 50.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

#### 50.6.1. Action items, Action Owners, Tracking, Checklist etc.

51. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was resolved and recommendations for testing.
- References

#### 52. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples

  - 1 A critical incident with very high impact
  - A

customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted 3 A minor incident with low impact • A minor inconvenience to customers, workaround available In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs Blocker — Highest priority. Indicates that this issue takes precedence over all others. Critical — Indicates that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority. 53. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 54. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process. 55. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 56. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project. 57. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester

and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

57.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

57.2. Defect States

#1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.

#2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.

#3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

y) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.

z) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.

aa) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.

bb) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".

#4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".

#5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".

#6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.

#7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.

#8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.

#9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

57.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

58. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

59. Defect Management

59.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier.

Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

59.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

59.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

59.2. Testing stages and defects Responsibilities SL No. Testing Stage Environment Defect Owner Defect Retested By Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team 4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager 59.3. Defect Triage Calls Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

59.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

59.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

59.4. Defect Metrics Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

59.5. Defect Reporting Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team

checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

59.5.1. Defect Trends by Release o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

59.5.2. Defects Density o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

59.5.3. Defect Fix Rate per Release o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

59.5.4. Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

59.5.5. Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish. Follow the steps below to create a fishbone diagram:

- (25) Write the problem at the head of the fish.
- (26) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
- (27) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
- (28) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

59.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea

regarding the correctness and/or the look and-feel of the system. #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects. #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

59.6.1. Action items, Action Owners, Tracking, Checklist etc.

60. Defect Data • Name of the Person • Types of Testing • Problem Summary • Detailed Description of Defect. • Steps to Reproduce • Life Cycle Phase • Work product where Defect was introduced. • Severity and Priority • Subsystem or Component where the Defect is introduced. • Project Activity occurs when the Defect is introduced. • Identification Method • Type of Defect • Projects and Products in which problems exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References

61. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples

1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss

2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted

3 A minor incident with low impact • A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

Blocker — Highest priority. Indicates that this issue takes precedence over all others. Critical — Indicates that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority.

62. Defect Fix timeline

Severity Response Time Resolution Time 1 2 3 4 63. Conclusion

This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

64. What is A Defect?

A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

65. Defect Management Life Cycle

When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is

different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

**66. Defect Life Cycle in Detail**

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

**66.1. Defect Workflow**

The diagram below shows the actual life cycle of a defect.

**66.2. Defect States**

- #1) **New:** This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) **Assigned:** In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) **Open:** Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.
- cc)  
**Rejected:** If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- dd)  
**Duplicate:** If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.
- ee)  
**Deferred:** If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.
- ff)  
**Not a Bug:** If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".
- #4) **Fixed:** When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".
- #5) **Pending Retest:** After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".
- #6) **Retest:** At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.
- #7) **Reopen:** If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.
- #8) **Verified:** If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.
- #9) **Closed:** When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

**66.3. Guidelines for Implementing a Defect Life Cycle**

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is

very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).

- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 67. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 68. Defect Management

#### 68.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

#### 68.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

#### 68.1.2. Why is Defect Tracking Necessary?

- A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- Monitors the bugs that had already been resolved, thereby saving time.
- To ensure that right defects are being worked on.

#### 68.2. Testing stages and defects Responsibilities

SL No. Testing Stage Environment Defect Owner Defect Retested By

Defect Tracking	By 1 Unit Testing (UT)	DEV DEV Team	Dev Team	2 System Testing (ST)	ST ST
Team ST	Team ST	Team 3	System Integration Testing (SIT)	SIT SIT Team	SIT Team SIT Team
4 User Acceptance Testing (UAT)	UAT UAT Team	UAT Team	Test Manager	68.3. Defect Triage Calls	

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

#### 68.3.1. Thumb rules of Triaging

- All reported defects have been reviewed
- All accepted defects have been prioritized
- All accepted defects have severity attached
- All rejected defects should have reasonable explanations for the testing team
- Every defect has been assigned to an appropriate owner, individual or team
- Root cause analysis for every accepted defect has been done

#### 68.3.2. How does Defect Triaging work?

The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach

priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

#### 68.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

#### 68.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

##### 68.5.1. Defect Trends by Release

o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

##### 68.5.2. Defects Density

o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

##### 68.5.3. Defect Fix Rate per Release

o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

##### 68.5.4. Defect Aging

o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

##### 68.5.5. Defect Analysis through RCA

o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
- Follow the steps below to create a fishbone diagram:
  - (29) Write the problem at the head of the fish.
  - (30) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
  - (31) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
  - (32) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe

the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

### 6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

#### 6.8.6.1. Action items, Action Owners, Tracking, Checklist etc.

#### 6.9. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was resolved and recommendations for testing.
- References

#### 70. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first?

Severity Description Examples

- 1 A critical incident with very high impact
- A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss

- 2 A major incident with significant impact
- A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted

- 3 A minor incident with low impact
- A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

#### 71. Defect Fix timeline

Severity Response Time Resolution Time

1	2	3
4	72. Conclusion	This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and

development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

73. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

74. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

75. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

75.1. Defect Workflow The diagram below shows the actual life cycle of a defect.

75.2. Defect States #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.

#2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.

#3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

gg) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.

hh) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of

the defect is changed to 'Duplicate' by the developer. ii) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.

**Defect Management Process** Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

1. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

2. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

3. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

3.1. Defect Workflow The diagram below shows the actual life cycle of a defect.

3.2. Defect States #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.

#2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.

#3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the

developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

a) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.

b) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.

c) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.

d) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".

#4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".

#5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".

#6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.

#7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.

#8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.

#9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 3.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 4. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 5. Defect Management

#### 5.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

##### 5.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

##### 5.1.2. Why is Defect Tracking Necessary?

- o

A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.

- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

### 5.2. Testing stages and defects

Responsibilities SL No. Testing Stage Environment Defect Owner Defect Retested By Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team 4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager

### 5.3. Defect Triage Calls

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

#### 5.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

#### 5.3.2. How does Defect Triaging work?

The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

### 5.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

### 5.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

#### 5.5.1. Defect Trends by Release

- o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

#### 5.5.2. Defects Density

- o Defect density can be defined as the

number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

- o Defect Density = Total Defect/Size of the Release
- 5.5.3. Defect Fix Rate per Release
- o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .
- o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

5.5.4. Defect Aging

- o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

5.5.5. Defect Analysis through RCA

- o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.
- o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.

Follow the steps below to create a fishbone diagram:

- (1) Write the problem at the head of the fish.
- (2) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
- (3) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
- (4) Extend the causes to secondary, tertiary, and more levels as applicable.

o Pareto charts to do the affinity analysis

- A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement
- Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit.

The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

5.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

5.6.1. Action items, Action Owners, Tracking, Checklist etc.

6. Defect Data
  - Name of the Person
  - Types of Testing
  - Problem Summary
  - Detailed Description of Defect.
  - Steps to Reproduce
  - Life Cycle Phase
  - Work product where Defect was introduced.
  - Severity and Priority
  - Subsystem or Component where the Defect is introduced.
  - Project Activity occurs when the Defect is introduced.

Identification Method • Type of Defect • Projects and Products in which problems exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.

- Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References

7. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples

- 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss
- 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted
- 3 A minor incident with low impact • A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

8. Defect Fix timeline

Severity	Response Time	Resolution Time
1	2	3
4	9	

Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

10. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

11. Defect Management Life Cycle

When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the

above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

### 12. Defect Life Cycle in Detail

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

#### 12.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

#### 12.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.
- e) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- f) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.
- g) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.
- h) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".
- #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".
- #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".
- #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.
- #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.
- #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.
- #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

#### 12.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects

and thus, in the workflow of the Defect Life Cycle.

### 13. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 14. Defect Management

#### 14.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

#### 14.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

#### 14.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

#### 14.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Defect Retested By
1	Unit Testing (UT)	DEV	DEV Team	Dev Team
2	System Testing (ST)	ST	ST Team	ST Team
3	System Integration Testing (SIT)	SIT	SIT Team	SIT Team
4	User Acceptance Testing (UAT)	UAT	UAT Team	UAT Team
			Test Manager	

#### 14.3. Defect Triage

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

#### 14.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

#### 14.3.2. How does Defect Triaging work?

The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which

ensures that chances of getting a similar defect are significantly reduced.

#### 14.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

#### 14.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

##### 14.5.1. Defect Trends by Release

- The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

##### 14.5.2. Defects Density

- Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

##### 14.5.3. Defect Fix Rate per Release

- Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

##### 14.5.4. Defect Aging

- Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

##### 14.5.5. Defect Analysis through RCA

- RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.
- Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.
- Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish. Follow the steps below to create a fishbone diagram:
  - (5) Write the problem at the head of the fish.
  - (6) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
  - (7) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
  - (8) Extend the causes to secondary, tertiary, and more levels as applicable.
- Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:
  - Identifying and prioritizing problems and tasks.
  - Helping people to organize their workloads more effectively.
  - Improving productivity.
  - Improving profitability.

##### Pareto Analysis Steps

1. Define the problem or goal.
2. Collect data on the frequency of each problem or cause.
3. Rank the causes based on their frequency.
4. Create a Pareto chart to visualize the data.
5. Identify the most significant causes (the "vital few").
6. Develop and implement solutions for the top causes.
7. Monitor the results and make adjustments as needed.

Identify and List Problems 2. Identify the Root Cause of Each Problem 3. Score Problems 4. Group Problems Together 5. Add up Scores for Each Group 6. Take Action 14.6. Defect Prevention Processes & Improvements Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks: #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products. #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system. #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects. #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis 14.6.1. Action items, Action Owners, Tracking, Checklist etc. 15. Defect Data • Name of the Person • Types of Testing • Problem Summary • Detailed Description of Defect. • Steps to Reproduce • Life Cycle Phase • Work product where Defect was introduced. • Severity and Priority • Subsystem or Component where the Defect is introduced. • Project Activity occurs when the Defect is introduced. • Identification Method • Type of Defect • Projects and Products in which problems exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References 16. Defect Priority and Severity Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted 3 A minor incident with low impact • A minor inconvenience to customers, workaround available In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs Blocker — Highest priority. Indicates that this issue takes precedence over all others. Critical — Indicates that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority. 17. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 18. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process. 19. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an

application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 20.

**Defect Management Life Cycle** When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project. 21.

**Defect Life Cycle in Detail** The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again. 21.1. **Defect Workflow** The diagram below shows the actual life cycle of a defect. 21.2.

**Defect States #1) New:** This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle. #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer. #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason. i) **Rejected:** If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer. j) **Duplicate:** If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer. k) **Deferred:** If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'. l) **Not a Bug:** If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug". #4) **Fixed:** When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed". #5) **Pending Retest:** After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest". #6) **Retest:** At this point, the tester starts the task of

retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 21.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 22. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 23. Defect Management

#### 23.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

##### 23.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

##### 23.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

### 23.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Defect Retested By
1	Unit Testing (UT)	DEV	DEV Team	Dev Team
2	System Testing (ST)	ST	ST Team	ST Team
3	System Integration Testing (SIT)	SIT	SIT Team	SIT Team
4	User Acceptance Testing (UAT)	UAT	UAT Team	UAT Team

### 23.3. Defect Triage Calls

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing

them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

23.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

23.3.2. How does Defect Triaging work?

The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

23.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

23.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

23.5.1. Defect Trends by Release

- o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

23.5.2. Defects Density

- o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.
- o Defect Density = Total Defect/Size of the Release

23.5.3. Defect Fix Rate per Release

- o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .
- o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

23.5.4. Defect Aging

- o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

23.5.5. Defect Analysis through RCA

- o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.
- o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and

effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms. Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish. Follow the steps below to create a fishbone diagram: (9) Write the problem at the head of the fish. (10) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N] (11) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N. (12) Extend the causes to secondary, tertiary, and more levels as applicable.

o Pareto charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

### 23.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

#### 23.6.1. Action items, Action Owners, Tracking, Checklist etc.

24. Defect Data
  - Name of the Person
  - Types of Testing
  - Problem Summary
  - Detailed Description of Defect.
  - Steps to Reproduce
  - Life Cycle Phase
  - Work product where Defect was introduced.
  - Severity and Priority
  - Subsystem or Component where the Defect is introduced.
  - Project Activity occurs when the Defect is introduced.
  - Identification Method
  - Type of Defect
  - Projects and Products in which problems exist
  - Current Owner
  - Current State of the Report
  - Work product where Defect occurred.
  - Impact on Project
  - Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
  - Dates when various defect lifecycle phases occur.
  - Description of how the defect was resolved and recommendations for testing.
  - References
25. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples

- 1 A critical incident with very high impact
- A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss
- 2 A major incident with significant impact
- A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted

3

A minor incident with low impact • A minor inconvenience to customers, workaround available In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs Blocker — Highest priority. Indicates that this issue takes precedence over all others. Critical — Indicates that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority. 26. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 27. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process. 28. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 29. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project. 30. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again. 30.1. Defect Workflow The diagram below shows the actual life cycle of a defect. 30.2. Defect States #1) New: This is the first state of a defect in the Defect Life Cycle. When any

new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle. #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer. #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason. m) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer. n) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer. o) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'. p) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug". #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed". #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest". #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 30.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 31. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 32. Defect Management

#### 32.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

#### 32.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry

as it provides a faster method of correcting the defects o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction. o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects. o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

### 32.1.2. Why is Defect Tracking Necessary?

o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process. o Monitors the bugs that had already been resolved, thereby saving time. o To ensure that right defects are being worked on.

### 32.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Defect Retested By
1	Unit Testing (UT)	DEV	DEV Team	Dev Team
2	System Testing (ST)	ST	ST Team	ST Team
3	System Integration Testing (SIT)	SIT	SIT Team	SIT Team
4	User Acceptance Testing (UAT)	UAT	UAT Team	UAT Team
			Test Manager	

### 32.3. Defect Triage

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

#### 32.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

#### 32.3.2. How does Defect Triaging work?

The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

### 32.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

### 32.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management

process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

32.5.1. Defect Trends by Release o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

32.5.2. Defects Density o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

32.5.3. Defect Fix Rate per Release o Defect removal efficiency =  $(1 - (\text{total defects caught by customer/ total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

32.5.4. Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

32.5.5. Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
- Follow the steps below to create a fishbone diagram:
  - (13) Write the problem at the head of the fish.
  - (14) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
  - (15) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
  - (16) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement.

Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit.

The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes.

Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

32.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects.

Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes

two major approaches: I) Pareto Analysis and II) Fishbone Analysis

### 32.6.1. Action items, Action Owners, Tracking, Checklist etc.

33. Defect Data • Name of the Person • Types of Testing • Problem Summary • Detailed Description of Defect. • Steps to Reproduce • Life Cycle Phase • Work product where Defect was introduced. • Severity and Priority • Subsystem or Component where the Defect is introduced. • Project Activity occurs when the Defect is introduced. • Identification Method • Type of Defect • Projects and Products in which problems exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References

### 34. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples

- 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached
- 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted
- 3 A minor incident with low impact • A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

### 35. Defect Fix timeline

Severity Response Time Resolution Time 1 2 3 4

### 36. Conclusion

This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

### 37. What is A Defect?

A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

### 38. Defect Management Life Cycle

When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the

projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

### 39. Defect Life Cycle in Detail

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

#### 39.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

#### 39.2. Defect States

#1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.

#2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.

#3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

q) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.

r) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.

s) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.

t) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".

#4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".

#5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".

#6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.

#7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.

#8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.

#9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 39.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual

who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.

- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

#### 40. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

#### 41. Defect Management

##### 41.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

##### 41.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

##### 41.1.2. Why is Defect Tracking Necessary?

- A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- Monitors the bugs that had already been resolved, thereby saving time.
- To ensure that right defects are being worked on.

##### 41.2. Testing stages and defects Responsibilities

SL No. Testing Stage Environment Defect Owner Defect Retested By Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team 4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager

##### 41.3. Defect Triage

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the “right” defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

##### 41.3.1. Thumb rules of Triaging

- All reported defects have been reviewed
- All accepted defects have been prioritized
- All accepted defects have severity attached
- All rejected defects should have reasonable explanations for the testing team
- Every defect has been assigned to an appropriate owner, individual or team
- Root cause analysis for every accepted defect has been done

##### 41.3.2. How does Defect Triaging work?

The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be

considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

#### 41.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

#### 41.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

##### 41.5.1. Defect Trends by Release

o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

##### 41.5.2. Defects Density

o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

##### 41.5.3. Defect Fix Rate per Release

o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

##### 41.5.4. Defect Aging

o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

##### 41.5.5. Defect Analysis through RCA

o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
- Follow the steps below to create a fishbone diagram:
  - (17) Write the problem at the head of the fish.
  - (18) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
  - (19) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
  - (20) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement. Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the

Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

#### 41.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

#### 41.6.1. Action items, Action Owners, Tracking, Checklist etc.

42. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was resolved and recommendations for testing.
- References

#### 43. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples

- 1 A critical incident with very high impact
- 2 A customer-facing service, like Jira Cloud, is down for all customers
- 3 Confidentiality or privacy is breached
- 4 Customer data loss

2 A major incident with significant impact

3 A customer-facing service is unavailable for a subset of customers

4 Core functionality is significantly impacted

3 A minor incident with low impact

4 A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

#### 44. Defect Fix timeline

Severity Response Time Resolution Time

1	2	3
4	45.	Conclusion

This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in

software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

46. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

47. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

48. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

48.1. Defect Workflow The diagram below shows the actual life cycle of a defect.

48.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.
- u) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- v) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.
- w) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.
- x) Not a Bug: If the defect

does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug". #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed". #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest". #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 48.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 49. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 50. Defect Management

#### 50.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

##### 50.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

##### 50.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

#### 50.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Defect Retested By
1	Unit Testing (UT)	DEV	DEV Team	Dev Team
2	System Testing (ST)	ST	ST Team	ST Team
3	System Integration Testing (SIT)	SIT	SIT Team	SIT Team
4	User Acceptance Testing (UAT)	UAT	UAT Team	UAT Team

#### 50.3. Defect Triage Calls

Defect triage is a process to prioritize defects based on severity, risk, frequency of

occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the “right” defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

50.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

50.3.2. How does Defect Triaging work?

The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

50.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

50.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

50.5.1. Defect Trends by Release

- o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

50.5.2. Defects Density

- o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.
- o Defect Density = Total Defect/Size of the Release

50.5.3. Defect Fix Rate per Release

- o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .
- o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

50.5.4. Defect Aging

- o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

50.5.5. Defect Analysis through RCA

- o RCA

(Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

- o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms. Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish. Follow the steps below to create a fishbone diagram: (21)Write the problem at the head of the fish. (22)Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N] (23)Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N. (24)Extend the causes to secondary, tertiary, and more levels as applicable.
- o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

#### 50.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

#### 50.6.1. Action items, Action Owners, Tracking, Checklist etc.

51. Defect Data
  - Name of the Person
  - Types of Testing
  - Problem Summary
  - Detailed Description of Defect
  - Steps to Reproduce
  - Life Cycle Phase
  - Work product where Defect was introduced.
  - Severity and Priority
  - Subsystem or Component where the Defect is introduced.
  - Project Activity occurs when the Defect is introduced.
  - Identification Method
  - Type of Defect
  - Projects and Products in which problems exist
  - Current Owner
  - Current State of the Report
  - Work product where Defect occurred.
  - Impact on Project
  - Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
  - Dates when various defect lifecycle phases occur.
  - Description of how the defect was resolved and recommendations for testing.
  - References
52. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing

a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted 3 A minor incident with low impact • A minor inconvenience to customers, workaround available In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs Blocker — Highest priority. Indicates that this issue takes precedence over all others. Critical — Indicates that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority. 53. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 54. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process. 55. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 56. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The

number of states that a defect goes through varies from project to project.

**57. Defect Life Cycle in Detail**

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

**57.1. Defect Workflow**

The diagram below shows the actual life cycle of a defect.

**57.2. Defect States**

- #1) **New:** This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) **Assigned:** In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) **Open:** Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.
- y) **Rejected:** If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- z) **Duplicate:** If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.
- aa) **Deferred:** If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.
- bb) **Not a Bug:** If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".
- #4) **Fixed:** When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".
- #5) **Pending Retest:** After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".
- #6) **Retest:** At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.
- #7) **Reopen:** If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.
- #8) **Verified:** If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.
- #9) **Closed:** When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

**57.3. Guidelines for Implementing a Defect Life Cycle**

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

**58. Goals of Defect Management Process (DMP)**

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

**59. Defect Management**

**59.1. Defect Tracking**

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and

business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

59.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

59.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

59.2. Testing stages and defects Responsibilities SL No. Testing Stage Environment Defect Owner Defect Retested By Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team 4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager 59.3. Defect Triage Calls Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

59.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

59.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

59.4. Defect Metrics Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA,

UAT or End-user 59.5. Defect Reporting Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

59.5.1. Defect Trends by Release o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

59.5.2. Defects Density o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

59.5.3. Defect Fix Rate per Release o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

59.5.4. Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

59.5.5. Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
- Follow the steps below to create a fishbone diagram:
  - (25) Write the problem at the head of the fish.
  - (26) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
  - (27) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
  - (28) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

59.6. Defect Prevention Processes & Improvements Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

#1) Review and Inspection: This

method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products. #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system. #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects. #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

59.6.1. Action items, Action Owners, Tracking, Checklist etc.

60. Defect Data • Name of the Person • Types of Testing • Problem Summary • Detailed Description of Defect. • Steps to Reproduce • Life Cycle Phase • Work product where Defect was introduced. • Severity and Priority • Subsystem or Component where the Defect is introduced. • Project Activity occurs when the Defect is introduced. • Identification Method • Type of Defect • Projects and Products in which problems exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References

61. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples

- 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss
- 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted
- 3 A minor incident with low impact • A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

62. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4

63. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

64. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

65. Defect Management Life Cycle When a system gives a different output other than the actual

business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

**66. Defect Life Cycle in Detail**

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

**66.1. Defect Workflow**

The diagram below shows the actual life cycle of a defect.

**66.2. Defect States #1) New:** This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.

**#2) Assigned:** In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.

**#3) Open:** Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

**cc)**

**Rejected:** If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.

**dd) Duplicate:** If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.

**ee) Deferred:** If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.

**ff) Not a Bug:** If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".

**#4) Fixed:** When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".

**#5) Pending Retest:** After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".

**#6) Retest:** At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.

**#7) Reopen:** If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.

**#8) Verified:** If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.

**#9) Closed:** When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields

where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter. 66.3. Guidelines for Implementing a Defect Life Cycle Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

67. Goals of Defect Management Process (DMP) Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

68. Defect Management 68.1. Defect Tracking In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

68.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

68.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

68.2. Testing stages and defects Responsibilities SL No. Testing Stage Environment Defect Owner Defect Retested By Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team 4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager

68.3. Defect Triage Calls Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

68.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

68.3.2. How does Defect Triaging work? The

defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

#### 68.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

#### 68.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

##### 68.5.1. Defect Trends by Release

o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

##### 68.5.2. Defects Density

o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

##### 68.5.3. Defect Fix Rate per Release

o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

##### 68.5.4. Defect Aging

o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

##### 68.5.5. Defect Analysis through RCA

o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
- Follow the steps below to create a fishbone diagram:
  - (29) Write the problem at the head of the fish.
  - (30) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
  - (31) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
  - (32) Extend the

causes to secondary, tertiary, and more levels as applicable.

- o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement. Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

68.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

68.6.1. Action items, Action Owners, Tracking, Checklist etc.

69. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was resolved and recommendations for testing.
- References

70. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples

- 1 A critical incident with very high impact
- A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss
- 2 A major incident with significant impact
- A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted
- 3 A minor incident with low impact
- A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

71. Defect Fix timeline

Severity Response Time Resolution Time

- 1
- 2
- 3
- 4
72. Conclusion

This was all there is to know about the Bug Life Cycle and how the testing

and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

73. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

74. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

75. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

75.1. Defect Workflow The diagram below shows the actual life cycle of a defect.

75.2. Defect States #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle. #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer. #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

gg)

Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer. hh) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer. ii) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.

**Defect Management Process** Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

1. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

2. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

3. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

3.1. Defect Workflow The diagram below shows the actual life cycle of a defect.

3.2. Defect States #1) New: This is the first state of a defect in the Defect

Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle. #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer. #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

a) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer. b) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer. c) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'. d) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug". #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed". #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest". #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 3.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 4. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 5. Defect Management

#### 5.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

##### 5.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a

faster method of correcting the defects o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction. o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects. o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

5.1.2.Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

5.2.Testing stages and defects Responsibilities SL No. Testing Stage Environment Defect Owner Defect Retested By Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team 4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager

5.3.Defect Triage Calls

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

5.3.1.Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

5.3.2.How does Defect Triaging work?

The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

5.4.Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

5.5.Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to

support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

5.5.1. Defect Trends by Release o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

5.5.2. Defects Density o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

5.5.3. Defect Fix Rate per Release o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

5.5.4. Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

5.5.5. Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.

Follow the steps below to create a fishbone diagram:

- (1) Write the problem at the head of the fish.
- (2) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
- (3) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
- (4) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement

Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit.

The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes.

Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

5.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects.

Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes

two major approaches: I) Pareto Analysis and II) Fishbone Analysis

5.6.1. Action items, Action Owners, Tracking, Checklist etc.

6. Defect Data • Name of the Person • Types of Testing • Problem Summary • Detailed Description of Defect. • Steps to Reproduce • Life Cycle Phase • Work product where Defect was introduced. • Severity and Priority • Subsystem or Component where the Defect is introduced. • Project Activity occurs when the Defect is introduced. • Identification Method • Type of Defect • Projects and Products in which problems exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.

• Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References

7. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples

1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss

2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted

3 A minor incident with low impact • A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

Blocker — Highest priority. Indicates that this issue takes precedence over all others.

Critical — Indicates that this issue is causing a problem and requires urgent attention.

Major — Indicates that this issue has a significant impact.

Minor — Indicates that this issue has a relatively minor impact.

Trivial — Lowest priority.

8. Defect Fix timeline

Severity Response Time Resolution Time 1 2 3

4 9. Conclusion

This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free.

The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process

Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

10. What is A Defect?

A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

11. Defect Management Life Cycle

When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to

know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

### 12. Defect Life Cycle in Detail

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

#### 12.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

#### 12.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.
- e) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- f) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.
- g) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.
- h) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".
- #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".
- #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".
- #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.
- #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.
- #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.
- #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 12.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her

responsibility very clearly for better results. • Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily. • The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 13. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 14. Defect Management

#### 14.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

##### 14.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

##### 14.1.2. Why is Defect Tracking Necessary?

- A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- Monitors the bugs that had already been resolved, thereby saving time.
- To ensure that right defects are being worked on.

#### 14.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Defect Retested By
1	Unit Testing (UT)	DEV	DEV Team	Dev Team
2	System Testing (ST)	ST	ST Team	ST Team
3	System Integration Testing (SIT)	SIT	SIT Team	SIT Team
4	User Acceptance Testing (UAT)	UAT	UAT Team	UAT Team

#### 14.3. Defect Triage

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

##### 14.3.1. Thumb rules of Triaging

- All reported defects have been reviewed
- All accepted defects have been prioritized
- All accepted defects have severity attached
- All rejected defects should have reasonable explanations for the testing team
- Every defect has been assigned to an appropriate owner, individual or team
- Root cause analysis for every accepted defect has been done

##### 14.3.2. How does Defect Triaging work?

The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time

sensitivity for resolution • The complexity involved in the resolution • Business impact This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

#### 14.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

#### 14.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

##### 14.5.1. Defect Trends by Release

- o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

##### 14.5.2. Defects Density

- o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.
- o Defect Density = Total Defect/Size of the Release

##### 14.5.3. Defect Fix Rate per Release

- o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .
- o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

##### 14.5.4. Defect Aging

- o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

##### 14.5.5. Defect Analysis through RCA

- o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.
- o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.
- Steps to create a Fishbone Diagram:
  - Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
  - Follow the steps below to create a fishbone diagram:
  - (5) Write the problem at the head of the fish.
  - (6) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
  - (7) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
  - (8) Extend the causes to secondary, tertiary, and more levels as applicable.
- o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent

of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

#### 14.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

#### 14.6.1. Action items, Action Owners, Tracking, Checklist etc.

15. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was resolved and recommendations for testing.
- References

16. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first?

Severity Description Examples

- 1 A critical incident with very high impact
- A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss

- 2 A major incident with significant impact
- A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted

- 3 A minor incident with low impact
- A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

17. Defect Fix timeline

Severity Response Time Resolution Time

Severity	Response Time	Resolution Time
1	1	2
2	2	3
3	4	8

18. Conclusion

This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention

is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

19. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

20. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

21. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

21.1. Defect Workflow The diagram below shows the actual life cycle of a defect.

21.2. Defect States #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.

#2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.

#3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

i) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.

j) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.

k) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.

l) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect

gets changed to "Not a Bug". #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed". #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest". #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 21.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 22. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

### 23. Defect Management

#### 23.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

##### 23.1.1. Objectives of Defect Tracking

A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

##### 23.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

### 23.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Defect Retested By
1	Unit Testing (UT)	DEV	DEV Team	Dev Team
2	System Testing (ST)	ST	ST Team	ST Team
3	System Integration Testing (SIT)	SIT	SIT Team	SIT Team
4	User Acceptance Testing (UAT)	UAT	UAT Team	UAT Team

### 23.3. Defect Triage

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more

evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

23.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

23.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

23.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

23.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

23.5.1. Defect Trends by Release

- o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

23.5.2. Defects Density

- o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.
- o Defect Density = Total Defect/Size of the Release

23.5.3. Defect Fix Rate per Release

- o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .
- o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

23.5.4. Defect Aging

- o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

23.5.5. Defect Analysis through RCA

- o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a

Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

- o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.
- Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish. Follow the steps below to create a fishbone diagram:

  - (9) Write the problem at the head of the fish.
  - (10) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
  - (11) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
  - (12) Extend the causes to secondary, tertiary, and more levels as applicable.

- o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement

Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit.

The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes.

Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

### 23.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects.

Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

#### 23.6.1. Action items, Action Owners, Tracking, Checklist etc.

24. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was resolved and recommendations for testing.
- References

25. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand,

is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted 3 A minor incident with low impact • A minor inconvenience to customers, workaround available In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs Blocker — Highest priority. Indicates that this issue takes precedence over all others. Critical — Indicates that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority. 26. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 27. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process. 28. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 29. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project. 30. Defect Life

Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

### 30.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

30.2. Defect States #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle. #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer. #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

m) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.

n) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.

o) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.

p) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug". #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".

#5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".

#6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.

#7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.

#8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.

#9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

### 30.3. Guidelines for Implementing a Defect Life Cycle

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).

- Defect Life Cycle should be properly documented to avoid any confusion in the future.

- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.

- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.

- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

### 31. Goals of Defect Management Process (DMP)

Given below are the various goals of this process:

- Prevent the Defect

- Early Detection

- Minimize the impact

- Resolution of the Defect

- Process improvement

### 32. Defect Management

#### 32.1. Defect Tracking

In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing,

evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

32.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

32.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

32.2. Testing stages and defects Responsibilities SL No. Testing Stage Environment Defect Owner Defect Retested By Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team 4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager 32.3. Defect Triage Calls

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

32.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

32.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

32.4. Defect Metrics Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

32.5. Defect Reporting Defect Reporting in software testing is a process in

which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

32.5.1. Defect Trends by Release o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

32.5.2. Defects Density o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

32.5.3. Defect Fix Rate per Release o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

32.5.4. Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

32.5.5. Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish. Follow the steps below to create a fishbone diagram:

- (13) Write the problem at the head of the fish.
- (14) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
- (15) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
- (16) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

32.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

#1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and

inspection of all work products. #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system. #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects. #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

32.6.1. Action items, Action Owners, Tracking, Checklist etc.

33. Defect Data • Name of the Person • Types of Testing • Problem Summary • Detailed Description of Defect. • Steps to Reproduce • Life Cycle Phase • Work product where Defect was introduced. • Severity and Priority • Subsystem or Component where the Defect is introduced. • Project Activity occurs when the Defect is introduced. • Identification Method • Type of Defect • Projects and Products in which problems exist • Current Owner • Current State of the Report • Work product where Defect occurred. • Impact on Project • Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References

34. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples

- 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss
- 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted
- 3 A minor incident with low impact • A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

Blocker — Highest priority. Indicates that this issue takes precedence over all others. Critical — Indicates that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority.

35. Defect Fix timeline

Severity Response Time Resolution Time 1 2 3 4 36. Conclusion

This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process

Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

37. What is A Defect?

A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

38. Defect Management Life Cycle

When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business

requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

### 39. Defect Life Cycle in Detail

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

#### 39.1. Defect Workflow

The diagram below shows the actual life cycle of a defect.

#### 39.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.
- q) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- r) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.
- s) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.
- t) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".
- #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".
- #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".
- #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.
- #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.
- #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.
- #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase

and Reporter. 39.3. Guidelines for Implementing a Defect Life Cycle Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

40. Goals of Defect Management Process (DMP) Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

41. Defect Management 41.1. Defect Tracking In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

41.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

41.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

41.2. Testing stages and defects Responsibilities SL No. Testing Stage Environment Defect Owner Defect Retested By Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team

4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager 41.3. Defect Triage Calls Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

41.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

41.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes

stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

#### 41.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

#### 41.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

##### 41.5.1. Defect Trends by Release

The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

##### 41.5.2. Defects Density

o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

##### 41.5.3. Defect Fix Rate per Release

o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

##### 41.5.4. Defect Aging

o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

##### 41.5.5. Defect Analysis through RCA

o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
- Follow the steps below to create a fishbone diagram:

- (17) Write the problem at the head of the fish.
- (18) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
- (19) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
- (20) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity

analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

#### 41.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

#### 41.6.1. Action items, Action Owners, Tracking, Checklist etc.

42. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was resolved and recommendations for testing.
- References

#### 43. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first?

Severity Description Examples

- 1 A critical incident with very high impact
- A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss

- 2 A major incident with significant impact
- A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted

- 3 A minor incident with low impact
- A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

- Blocker — Highest priority. Indicates that this issue takes precedence over all others.
- Critical — Indicates that this issue is causing a problem and requires urgent attention.
- Major — Indicates that this issue has a significant impact.
- Minor — Indicates that this issue has a relatively minor impact.
- Trivial — Lowest priority.

#### 44. Defect Fix timeline

Severity Response Time Resolution Time

- 1
- 2
- 3

#### 45. Conclusion

This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-

free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

46. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

47. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

48. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

48.1. Defect Workflow The diagram below shows the actual life cycle of a defect.

48.2. Defect States

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.
- u) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as

"Rejected" by the developer. v) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer. w) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'. x) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug". #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed". #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest". #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

48.3. Guidelines for Implementing a Defect Life Cycle Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

49. Goals of Defect Management Process (DMP) Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

50. Defect Management

50.1. Defect Tracking In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

50.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

50.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

50.2. Testing stages and

defects Responsibilities SL No. Testing Stage Environment Defect Owner Defect Retested By Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team 4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager 50.3. Defect Triage Calls Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test. 50.3.1. Thumb rules of Triaging o All reported defects have been reviewed o All accepted defects have been prioritized o All accepted defects have severity attached o All rejected defects should have reasonable explanations for the testing team o Every defect has been assigned to an appropriate owner, individual or team o Root cause analysis for every accepted defect has been done 50.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are: • The validity of the defect • Time sensitivity for resolution • The complexity involved in the resolution • Business impact This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced. 50.4. Defect Metrics Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be: • Defect distribution by cause • Defect distribution by feature/functional area • Defect distribution by Severity • Defect distribution by Priority • Defect distribution by type • Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user 50.5. Defect Reporting Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them. 50.5.1. Defect Trends by Release o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed. 50.5.2. Defects Density o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software. o Defect Density = Total Defect/Size of the Release 50.5.3. Defect Fix Rate per Release o Defect removal efficiency = (1 - (total defects

caught by customer/ total number of defects)) x 100. o Residual defect density = (total number of defects found by a customer)/ (Total number of defects including customer found defects) x 100.

50.5.4. Defect Aging o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

50.5.5. Defect Analysis through RCA o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram: Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish. Follow the steps below to create a fishbone diagram:

- (21) Write the problem at the head of the fish.
- (22) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
- (23) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
- (24) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

50.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

50.6.1. Action items, Action Owners, Tracking, Checklist etc.

51. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the

defect. • Dates when various defect lifecycle phases occur. • Description of how the defect was resolved and recommendations for testing. • References 52. Defect Priority and Severity Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first? Severity Description Examples 1 A critical incident with very high impact • A customer-facing service, like Jira Cloud, is down for all customers • Confidentiality or privacy is breached • Customer data loss 2 A major incident with significant impact • A customer-facing service is unavailable for a subset of customers • Core functionality is significantly impacted 3 A minor incident with low impact • A minor inconvenience to customers, workaround available In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs Blocker — Highest priority. Indicates that this issue takes precedence over all others. Critical — Indicates that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority. 53. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4 54. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process. 55. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 56. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect

Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

**57. Defect Life Cycle in Detail**

The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

**57.1. Defect Workflow**

The diagram below shows the actual life cycle of a defect.

**57.2. Defect States**

- #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.
- #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer.
- #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.
- y) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer.
- z) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer.
- aa) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.
- bb) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug".
- #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed".
- #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest".
- #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.
- #7) Reopen: If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'.
- #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'.
- #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed".

The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

**57.3. Guidelines for Implementing a Defect Life Cycle**

Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

**58. Goals of Defect Management Process (DMP)**

Given below are the various goals of this process:

- Prevent the Defect
- Early Detection

- Minimize the impact • Resolution of the Defect • Process improvement

59.1. Defect Tracking In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

59.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects
 

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

59.1.2. Why is Defect Tracking Necessary?
 

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

59.2. Testing stages and defects Responsibilities SL No. Testing Stage Environment Defect Owner Defect Retested By

Defect Tracking By 1 Unit Testing (UT) DEV DEV Team Dev Team 2 System Testing (ST) ST ST

Team ST Team ST Team 3 System Integration Testing (SIT) SIT SIT Team SIT Team SIT Team

4 User Acceptance Testing (UAT) UAT UAT Team UAT Team Test Manager

59.3. Defect Triage Calls Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects identified with every test.

59.3.1. Thumb rules of Triaging
 

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team

Every defect has been assigned to an appropriate owner, individual or team
 

- o Root cause analysis for every accepted defect has been done

59.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

59.4. Defect Metrics Software quality assurance metrics must be used to track defects and structure the process of

their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

59.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

59.5.1. Defect Trends by Release

o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

59.5.2. Defects Density

o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.

o Defect Density = Total Defect/Size of the Release

59.5.3. Defect Fix Rate per Release

o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .

o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

59.5.4. Defect Aging

o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

59.5.5. Defect Analysis through RCA

o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.

o Fishbone Diagram

A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- Fishbone diagram resembles the skeleton of a fish with the problem forming the head of fish and causes forming the spine and bones of the fish.
- Follow the steps below to create a fishbone diagram:
  - (25) Write the problem at the head of the fish.
  - (26) Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N]
  - (27) Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N.
  - (28) Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis

A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement.

Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit.

The 80/20 Rule

Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes.

Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

59.6. Defect

**Prevention Processes & Improvements** Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches:
  - I) Pareto Analysis
  - II) Fishbone Analysis

59.6.1. Action items, Action Owners, Tracking, Checklist etc.

60. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was resolved and recommendations for testing.
- References

61. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first?

Severity Description Examples

- 1 A critical incident with very high impact
- A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss
- 2 A major incident with significant impact
- A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted
- 3 A minor incident with low impact
- A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

Blocker — Highest priority. Indicates that this issue takes precedence over all others.

Critical — Indicates that this issue is causing a problem and requires urgent attention.

Major — Indicates that this issue has a significant impact.

Minor — Indicates that this issue has a relatively minor impact.

Trivial — Lowest priority.

62. Defect Fix timeline

Severity Response Time Resolution Time

Severity	Response Time	Resolution Time
1	1	2
2	2	3
3	4	6

63. Conclusion

This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

64. What is A Defect?

A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an

application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect. 65.

**Defect Management Life Cycle** When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project. 66. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again. 66.1. Defect Workflow The diagram below shows the actual life cycle of a defect. 66.2. Defect States #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle. #2) Assigned: In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer. #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason. cc) Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer. dd) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer. ee) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'. ff) Not a Bug: If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to "Not a Bug". #4) Fixed: When the developer finishes the task of fixing a defect by making the required changes then they can mark the status of the defect as "Fixed". #5) Pending Retest: After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in "Pending Retest". #6) Retest: At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not. #7) Reopen: If any issue persists in the defect, then it will be

assigned to the developer again for testing and the status of the defect gets changed to 'Reopen'. #8) Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to 'Verified'. #9) Closed: When the defect does not exist any longer, then the tester changes the status of the defect to "Closed". The mandatory fields where a tester logs any new bug are Issue type, Summary, Linked Issues, Detected in Phase and Reporter.

66.3. Guidelines for Implementing a Defect Life Cycle Some important guidelines can be adopted before starting to work with the Defect Life Cycle. They are as follows:

- It is very important that before starting to work on the Defect Life Cycle, the whole team clearly understands the different states of a defect (discussed above).
- Defect Life Cycle should be properly documented to avoid any confusion in the future.
- Make sure that each individual who has been assigned any task related to the Defect Life Cycle should understand his/her responsibility very clearly for better results.
- Each individual who is changing the status of a defect should be properly aware of that status and should provide enough details about the status and the reason for putting that status so that everyone who is working on that particular defect can understand the reason of such a status of a defect very easily.
- The defect tracking tool should be handled with care to maintain consistency among the defects and thus, in the workflow of the Defect Life Cycle.

67. Goals of Defect Management Process (DMP) Given below are the various goals of this process:

- Prevent the Defect
- Early Detection
- Minimize the impact
- Resolution of the Defect
- Process improvement

68. Defect Management

68.1. Defect Tracking In software engineering, defect tracking is the process of tracking the logged defects in a product or application from beginning to closure (by inspection, testing, or recording feedback from customers), and making new versions of the product that fix the defects. It is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier. Defects are tracked based on various parameters such as:

- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status
- Reason

68.1.1. Objectives of Defect Tracking A defect tracker has revolutionized the software industry as it provides a faster method of correcting the defects

- o A defect tracker keeps track of all the defects. So that the management may not miss any defect from correction.
- o Due to these tracked defects, the most correct methods are adopted and preventive measures are taken to avoid further defects.
- o It saves the time of the engineers thus helping in the fast delivery of the work. Also, it enhances the efficient work delivered.

68.1.2. Why is Defect Tracking Necessary?

- o A defect tracker keeps a record of all the defects so that none of the errors is missed for evaluation process.
- o Monitors the bugs that had already been resolved, thereby saving time.
- o To ensure that right defects are being worked on.

68.2. Testing stages and defects Responsibilities

SL No.	Testing Stage	Environment	Defect Owner	Defect Retested By
1	Unit Testing (UT)	DEV	DEV Team	Dev Team
2	System Testing (ST)	ST	ST Team	ST Team
3	System Integration Testing (SIT)	SIT	SIT Team	SIT Team
4	User Acceptance Testing (UAT)	UAT	UAT Team	Test Manager

68.3. Defect Triage Calls

Defect triage is a process to prioritize defects based on severity, risk, frequency of occurrence. Though triaging can be implemented on any sized project, its benefits are more evident and tangible in large-sized projects. The list of defects could be proportionately big for large-sized projects. Alternatively, in a highly agile project that is spinning up, triage can help focus the team on the "right" defects. Either way, the triage process identifies defects that need immediate attention, while some may be deferred. The triaging mechanism helps in preparing a process for testers and developers to fix as many defects as possible by prioritizing them based on parameters identified and fixed by the team. Ideally, every test cycle should have regular triage sessions. Frequency can, however, depend on the number of defects

identified with every test.

68.3.1. Thumb rules of Triaging

- o All reported defects have been reviewed
- o All accepted defects have been prioritized
- o All accepted defects have severity attached
- o All rejected defects should have reasonable explanations for the testing team
- o Every defect has been assigned to an appropriate owner, individual or team
- o Root cause analysis for every accepted defect has been done

68.3.2. How does Defect Triaging work? The defect triage process involves holding a session / triage call with a triage team, which includes stakeholders like Product Manager, Testing Manager/Lead, Development Manager/Lead, and Business Analysts. The goal of this team is to evaluate the defects, assess them, and attach priorities and severity level. Priorities correspond to business perspective and severity corresponds to technicalities. Many times, a few defects may be considered trivial and rejected at this stage. Accepted defects are prioritized and assigned for resolution. Factors to be considered while evaluating and prioritizing the defects are:

- The validity of the defect
- Time sensitivity for resolution
- The complexity involved in the resolution
- Business impact

This process is not just about attaching severity and priority to the defects. It also provides all relevant information required to track, replicate, and fix them. The invalid defects and the basis of their rejection are also recorded for reference purposes. Root cause analysis for every single defect is conducted. This analysis forms the basis for an improvement plan which ensures that chances of getting a similar defect are significantly reduced.

68.4. Defect Metrics

Software quality assurance metrics must be used to track defects and structure the process of their resolution. Since it is usually not possible to debug every defect in a single sprint, bugs have to be allocated by priority, severity, testers availability and numerous other parameters. Some useful defect distribution metrics would be:

- Defect distribution by cause
- Defect distribution by feature/functional area
- Defect distribution by Severity
- Defect distribution by Priority
- Defect distribution by type
- Defect distribution by tester (or tester type) – Dev, QA, UAT or End-user

68.5. Defect Reporting

Defect Reporting in software testing is a process in which test managers / lead prepares and sends the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail. The management board has the right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report to them the current defect situation to get feedback from them.

68.5.1. Defect Trends by Release

- o The Defect Trends report in Software Testing calculates total number of defects that the QA team has opened and closed over time. The reports are important for management to understand the overall trend at which the defects are processed.

68.5.2. Defects Density

- o Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software.
- o Defect Density = Total Defect/Size of the Release

68.5.3. Defect Fix Rate per Release

- o Defect removal efficiency =  $(1 - (\text{total defects caught by customer} / \text{total number of defects})) \times 100$ .
- o Residual defect density =  $(\text{total number of defects found by a customer}) / (\text{Total number of defects including customer found defects}) \times 100$ .

68.5.4. Defect Aging

- o Defect Age (in Time) is the difference in time between the date a defect is detected and the current date (if the defect is still open) or the date the defect was fixed (if the defect is already fixed).

68.5.5. Defect Analysis through RCA

- o RCA (Root Cause Analysis) is a structured and effective process to find the root cause of issues in a Software Project team. If performed systematically, it can improve the performance and quality of the deliverables and the processes, not only at the team level but also across the organization.
- o Fishbone Diagram to know the RCAs A cause-and-effect diagram, often called a "fishbone" diagram, can help in brainstorming to identify possible causes of a problem and in sorting ideas into useful categories. A fishbone diagram is a visual way to look at cause and effect. It allows you to get down to the real root cause of the issue rather than solving its symptoms.

Steps to create a Fishbone Diagram:

- o Fishbone diagram resembles the skeleton of a

fish with the problem forming the head of fish and causes forming the spine and bones of the fish. Follow the steps below to create a fishbone diagram: (29)Write the problem at the head of the fish. (30)Identify the category of causes and write at end of each bone [cause category 1, cause category 2 ..... cause category N] (31)Identify the primary causes under each category and mark it as primary cause 1, primary cause 2, primary cause N. (32)Extend the causes to secondary, tertiary, and more levels as applicable.

o Parato charts to do the affinity analysis A Pareto Chart is a graph that indicates the frequency of defects, as well as their cumulative impact. Pareto Charts are useful to find the defects to prioritize in order to observe the greatest overall improvement Pareto Analysis is a simple decision-making technique for assessing competing problems and measuring the impact of fixing them. This allows you to focus on solutions that will provide the most benefit. The 80/20 Rule Pareto Analysis uses the Pareto Principle – also known as the "80/20 Rule". The Pareto Principle states that 80 percent of a project's benefit comes from 20 percent of the work. Or, conversely, that 80 percent of problems can be traced back to 20 percent of causes. Pareto Analysis identifies the problem areas or tasks that will have the biggest payoff. The tool has several benefits, including:

- Identifying and prioritizing problems and tasks.
- Helping people to organize their workloads more effectively.
- Improving productivity.
- Improving profitability.

Pareto Analysis Steps

1. Identify and List Problems
2. Identify the Root Cause of Each Problem
3. Score Problems
4. Group Problems Together
5. Add up Scores for Each Group
6. Take Action

#### 68.6. Defect Prevention Processes & Improvements

Defect prevention is a framework and ongoing process of collecting the defect data, doing root cause analysis, determining and implementing the corrective actions and sharing the lessons learned to avoid future defects. Defect Prevention is a crucial step or activity in any software development process and as can be seen from the diagram below is pretty much half of our testing tasks:

- #1) Review and Inspection: This method includes the review by an individual team member (self-checking), peer reviews and inspection of all work products.
- #2) Walkthrough: This is more or less like a review but it's mostly related to comparing the system to the prototype which will give a better idea regarding the correctness and/or the look and-feel of the system.
- #3) Defect Logging and Documentation: This method provides some key information, arguments/parameters that can be used to support analyzing defects.
- #4) Root Cause Analysis: Root cause analysis includes two major approaches: I) Pareto Analysis and II) Fishbone Analysis

#### 68.6.1. Action items, Action Owners, Tracking, Checklist etc.

69. Defect Data

- Name of the Person
- Types of Testing
- Problem Summary
- Detailed Description of Defect.
- Steps to Reproduce
- Life Cycle Phase
- Work product where Defect was introduced.
- Severity and Priority
- Subsystem or Component where the Defect is introduced.
- Project Activity occurs when the Defect is introduced.
- Identification Method
- Type of Defect
- Projects and Products in which problems exist
- Current Owner
- Current State of the Report
- Work product where Defect occurred.
- Impact on Project
- Risk, loss, opportunity, and benefits associated with fixing or not fixing the defect.
- Dates when various defect lifecycle phases occur.
- Description of how the defect was resolved and recommendations for testing.
- References

#### 70. Defect Priority and Severity

Severity represents the overall effect of a particular bug on a system and Priority defines how quickly a bug needs to be fixed. Severity is a measurement of impact. How much impact does an incident have on users? Does it take down their whole system? Keep them from completing a vital task? Or perhaps just irritate them and make tasks harder? Priority, on the other hand, is a measurement of urgency. How quickly do we need to fix this issue? Which issue needs to be fixed first?

Severity Description Examples

- 1 A critical incident with very high impact
- A customer-facing service, like Jira Cloud, is down for all customers
- Confidentiality or privacy is breached
- Customer data loss

- 2 A major incident with significant impact
- A customer-facing service is unavailable for a subset of customers
- Core functionality is significantly impacted

- 3 A minor incident with low impact
- A minor inconvenience to customers, workaround available

In Jira, under priority field there are Blocker, Critical, Major, Minor and Trivial Bugs

Blocker —

Highest priority. Indicates that this issue takes precedence over all others. Critical — Indicates that this issue is causing a problem and requires urgent attention. Major — Indicates that this issue has a significant impact. Minor — Indicates that this issue has a relatively minor impact. Trivial — Lowest priority.

71. Defect Fix timeline Severity Response Time Resolution Time 1 2 3 4

72. Conclusion This was all there is to know about the Bug Life Cycle and how the testing and development teams work to make the application that reaches the customers, bug/defect-free. The Defect Management Process should be followed during the overall software development process and not only for specific testing or development activities. Testing and development teams work side by side to fix the issue so that the system can work in its best form. Defect Management Process Defect management process is mainly used to improve the efficiency of the software development projects. Software quality is an important aspect in software development which ensures quality software product is developed. Defect Prevention is much more efficient and effective in reducing the number of defects and also is very cost effective to fix the defects found during the early stage of the software process. The organizations conduct Defect Discovery, Defect Removal and then Process Improvement which is collectively known as a Defect Management Process.

73. What is A Defect? A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do thorough testing of an application to find as many defects as possible to ensure that a quality product will reach the customer. It is important to understand the defect life cycle before moving to the workflow and different states of the defect.

74. Defect Management Life Cycle When a system gives a different output other than the actual business requirement i.e. when there is a deviation from the actual or original business requirement then we can say that there is a defect in the system/software. When the testing team executes the test cases, they come across a situation where the actual test result is different from the expected result. This variation is termed as a Defect. Basically, a Software Defect is a condition which does not meet the software requirement. The defect is an error or a flaw which produces unexpected or incorrect behavior in the system. In order to handle the projects appropriately, we need to know how to deal with the development and release, but along with that we also need to know how to handle defects. Just imagine, what will happen if the testing team reports the defects verbally and the development team also updates the status of defect verbally? The process will be more complicated as these defects include all defects like actually fixed and working as expected, fixed but still not working, rejected, postponed and the process goes on. In the above case, as the numbers of defects get increased and communication is performed verbally, the situation will soon be very worse. In order to control and handle the defect effectively, we need a formal Defect Life Cycle. Defect Life Cycle ensures that the process is uniform and standardized. A defect attains different states in the life cycle. After a defect has been found, it goes through various stages during its lifetime and it is commonly known as Defect Life Cycle. Generally, Defect life cycle starts from the stage when the defect is found or raised by the testing team and ends when a defect is closed either by ensuring that it's not reproducible or rejected by a development team. The number of states that a defect goes through varies from project to project.

75. Defect Life Cycle in Detail The Defect Life Cycle is a cycle of defects from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect, assuring that it won't get reproduced again.

75.1. Defect Workflow The diagram below shows the actual life cycle of a defect.

75.2. Defect States #1) New: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a 'New' state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle. #2) Assigned: In this stage, a newly created

defect is assigned to the development team to work on the defect. This is assigned by the QA or test lead of the testing team to a developer. #3) Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the four states below namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason. gg)  
Rejected: If the defect is not considered a genuine defect by the developer, then it is marked as "Rejected" by the developer. hh) Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to 'Duplicate' by the developer. ii) Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, developer can change the status of the defect as 'Deferred'.