Application Defect Management

OVERVIEW

A Primer on Defect Management

Defects in software can occur in any phase of the software creation process. The earlier and better they are managed, the easier they are to correct.

Once reviewed and a sign off is provided, the project moves through the design and coding phases. Testers then start developing the test plan and test cases while the developers work on coding the product. Once the product or part of the product is reviewed and unit tested, it is moved to the test staging environment where testers can execute test cases to find defects. Let’s understand this process in detail.

What is a Defect?

A defect can be described as non-conformance to the requirements or deviation from the expected behavior. Thus, any behavior which does not meet the end user’s expectations can be classified as a defect.

Defects can be found throughout all phases of the software development life cycle. Hence, defect categorization is helpful to identify its type, impact, urgency and the cost to fix.

Types of Defects

There are different types of defects which can be found in any software application. However, its impact varies by the type.

UI Defects	UI Defects or User Interface defects are cosmetic defects. Some examples of UI defects are button controls overlapping caption text, misspelt text in the company logo, error messages shown in green color rather than red, etc.
Functional Defects	Every application has a specified happy path which ensures that the application is working as expected. However, when these functions do not perform as expected, it gives rise to functional defects. Some examples would be a user not being able to create an account, failed money transfer transaction on a banking website or flight booking website, etc.
Database Defects	Databases are an integral part of an application. It helps to store, manage, retrieve and validate data against user inputs. Data accuracy, security, and integrity are some factors that are tested during the testing phase. Some examples of database related defects are connection failures to backend databases.
Integration defects	Integration testing is testing of two or more components of the application to ensure that the data flow is accurate. Such defects can be found during the critical path testing, happy path workflows or during the negative testing as well.
Architectural Defects	An invalid design or architectural flaw leads to unwanted/undesirable effects which require more efforts to fix, as the impact is more on the application. Defects at the architectural level can only be detected after major components in the system are ready for testing and are caught usually during system integration testing. Thus, requiring a lot of re-work to fix them.

Legacy Defects	In the world of Agile where software products are developed and delivered in fast, iterative phases, sometimes we might end up reusing test cases which were designed for the previous version of the application and are irrelevant for the new functionality that has been developed. This might result in presenting a wrong or invalid picture of the state of the test execution phase. Test case management by tracking and maintaining appropriate versions by releases and scenarios is essential to avoid such defects. Test case coverage and review techniques can also be helpful to overcome this issue in the early phase of test planning cycle
Requirement Defects	The requirements specification document is an artifact created and maintained by the business analysts. This document describes every component of a page or software application in details. It describes what technology will be used in developing the software, critical path scenarios, page look and feel, functions of every page and objects such as buttons, links, etc. which provides more clarity for developers and testers to understand the workflows and write code and test cases respectively. During the requirement review phase, requirements are analyzed to find gaps and items with assumptions are flagged for further review. These gaps are then filled in future revisions of the requirements document before any development effort begins. It is also possible to find requirements related defects further down the development phase as the build process begins or even after testing begins if not all the gaps are accounted for.

Defect Logging and Tracking

It is necessary that the testing team logs the defects correctly on finding reproducible defects. This helps the developers in managing development efforts using the defect priority and to reduce the turnaround time by quickly debugging to find the root cause of the defects. So, it is of the utmost importance for each defect logged to be complete with information that would aid in defect resolution.

Defect Details

Listing here are all the details that are necessary while logging any new defect.

Field	Description
ID	An ID or identifier is usually created in a defect management system which is a sequential number for defects to be identified or tracked.
Description	Summarizes the behavior of the application at a very high level. One can also add details such as any constraints or desired user role with which you must reproduce the defect.
Version of the application tested	Providing the version of the application helps the developer in comparing codebase to see changes between working and defect producing code.
Steps to Reproduce	Describing detailed step by step actions to be performed to reproduce the defect is useful information for anyone who is trying to reproduce the defect. It saves time for both testers and developers to explain the defect and to retest the same after it's been fixed.
Test Data used	Certain defects are generated only with some records of data. Hence adding the details of the test data which is being used during testing a scenario is necessary.
Preconditions	Preconditions are necessary in the following cases where for e.g.:

	1 the user needs to have the application in an initial state or 2 it breaks only in particular browser or 3 version of the browser Hence adding these details makes it easier to reproduce the defect.
Environment details	Since the software goes through different testing phases and through different test environments such as pilot, test, beta and then finally production, it is necessary to add environment details
Date Created	As the name suggests its the date when the defect was found, and this field is usually generated by the defect management system automatically.
Created by	If the test team consists of multiple testers, it is useful to know who logged this defect.
Status	Knowing the status of the defect is useful for the team to decide the next steps to be actioned on the defect. This is explained in detail in the next section.
Date Closed	A system generated field automatically gets filled when the user changes the status of the defect as closed. It is useful information in cases where the tester must reopen or reference the same defect in the next release.
Severity	Severity can be defined as the impact on the system due to the introduction of the defect. It ranges from whether the defect can bring the entire system down or it can be a normal functionality drift.
Priority	Priority can be described as the urgency or a timeframe to resolve the defect. If it has an impact on a broader audience, then the probability to resolve it is likely to be soon or on an urgent basis. Also, higher severity defects do get higher priorities.
Expected	The expected behavior is the condition or the
Behavior	result we expect after performing the series of steps in the form of a test case. If it matches the requirement there are no issues however if there is any discrepancy with the requirement

	specified, then we need to add the details about what exactly we expected.
Actual Behavior	Actual behavior captures the current behavior of the application on performing the series of the test steps.
Artifacts	Artifacts are evidence that the testers can attach to the defect. It gives more clarity for developers to investigate the issue further. It can be test data used for testing, the application URL, logs, screenshots, actual error message displayed on the application, etc.

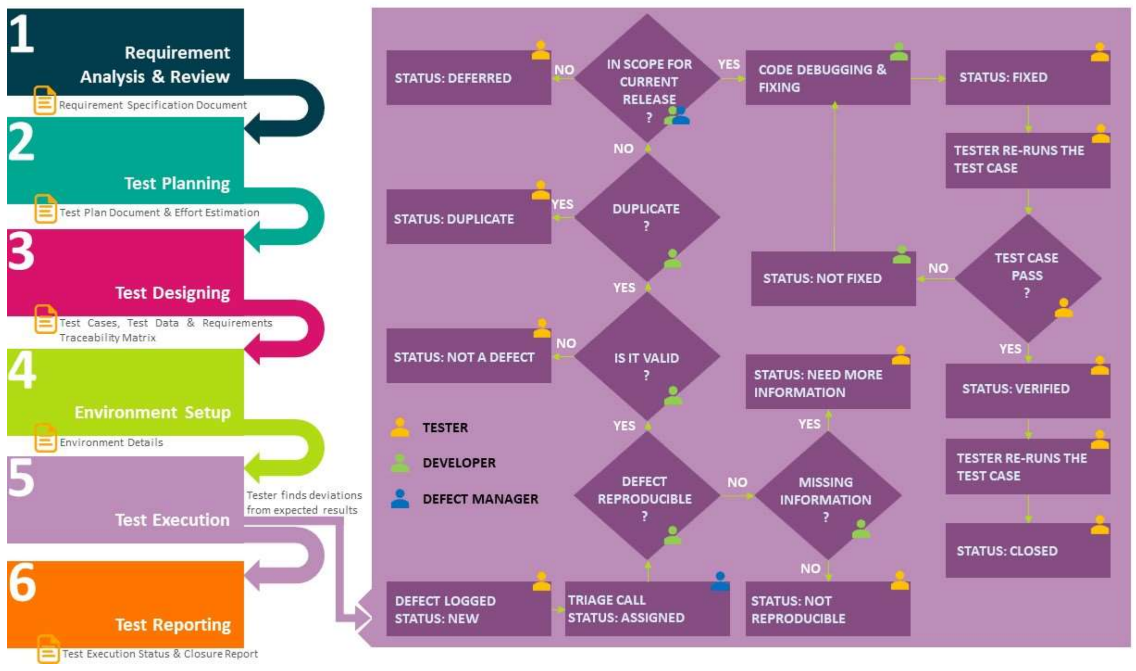
Defect Status

As the project progresses, each defect goes through several stages of development or resolution called the Defect Life Cycle. During this cycle, the status as well as the assignee (to whom the defect is assigned) keeps on changing. The table below shows the various states:

Status	Description
New	When the defect is created, it’s in the new status. Some systems also refer to this status as
	submitted. Since it has been just created, it sits with the reporter who logged this defect.
Assigned	Defect status turns into assigned when the team identifies the appropriate person to work on it.
Not a defect	The developer marks the defect to a “Not a defect” if the system behavior is as per expectations.
Fixed	When a developer identifies the root cause and makes necessary changes in the code, they make it available for testers to verify the changes by marking the defect status as fixed.
Not Fixed	Testers then look at the assigned defects and verify the new changes, but notices that it’s still breaking or not working as expected. At this point, the tester can change the defect status to not fixed and assign it back to the concerned developer.
Need more information	Sometimes the details added during the creation of defects are insufficient to dig deeper, and developers need more data to find the root cause. In such cases, they can mark the status of defect as needs more information.
Not reproducible	Sometimes a defect appears only once, and it becomes really challenging to reproduce it despite using all the specified details such as

	test data, browser, test environment, same version of the application, etc. In such cases, developers mark defects as not reproducible.
Deferred	If the defect is not in the scope of the current release, it can be marked as deferred to be fixed in the later releases.
Duplicate	If the same issue has been reported by the same or different reporter and it still has not been resolved and closed, it is marked as Duplicate or Existing.
Verified	Once developers fix the defect, they make it ready for testers to test. The tester then verifies the changes against the requirement and confirms that the behavior is as expected and if it conforms, they can mark it as Verified.
Closed	Once the defect has been verified and it generates the expected results, tester can mark the defect as Closed after all artifacts have been attached to the defect. Ideally, this should be the last status for a defect.
Reopen	If a defect which was found in the previous release or previous version of the test cycle is found again, testers might want to reopen existing closed defect. Such a defect goes through the different stages and can be marked as closed once the changes are verified and it works as expected.

As we have seen so far, the software development life cycle comprises of several steps, with each step producing artifacts, which acts as an input for its subsequent phases. For e.g., the requirement specification document is the artifact produced after the Requirement Analysis and Review phase and so on. Once the test execution phase begins, a new defect is logged, and the triage call is initiated. This process is illustrated below:


1 Requirement Analysis & Review
Requirement Specification Document
2 Test Planning
Test Plan Document & Effort Estimation
3 Test Designing
Test Cases, Test Data & Requirements Traceability Matrix
4 Environment Setup
Environment Details
5 Test Execution
Tester finds deviations from expected results
6 Test Reporting
Test Execution Status & Closure Report
STATUS: DEFERRED
NO
IN SCOPE FOR CURRENT RELEASE ?
YES
CODE DEBUGGING & FIXING
STATUS: FIXED
NO
TESTER RE-RUNS THE TEST CASE
STATUS: DUPLICATE
YES
DUPLICATE ?
YES
STATUS: NOT FIXED
NO
TEST CASE PASS ?
STATUS: NOT A DEFECT
NO
IS IT VALID ?
STATUS: NEED MORE INFORMATION
YES
STATUS: VERIFIED
YES
YES
TESTER
DEVELOPER
DEFECT MANAGER
DEFECT REPRODUCIBLE ?
NO
MISSING INFORMATION ?
TESTER RE-RUNS THE TEST CASE
STATUS: CLOSED
NO
DEFECT LOGGED STATUS: NEW
TRIAGE CALL STATUS: ASSIGNED
STATUS: NOT REPRODUCIBLE

Defect Triage and Assignment

During the test execution phase, project team members meet once a day to go over defects. The meeting is scheduled on a recurring basis until all the defects have been marked as deferred or closed. The Defect Manager usually conducts such meetings with the development and testing teams to discuss newly discovered defects, the number of test cases blocked or failed because of the defects, defects which worked as expected and closed, defects which have been fixed by developers and assigned back to the testing team and so on. Such meetings are called Defect Triage calls.

The defect manager considers various factors while assigning defects to the developers. Along with the priority and severity of the defect, the defect manager also checks the availability and expertise of the developers.

Priority of a defect has four different degrees which determine the requirement of how soon it needs to be resolved. Each of them has a general SLA (Service Level Agreement) timeline defined in the contract

Code	Type	Description
P0	Critical	Meaning it has the highest priority and we need to fix it immediately without any delays. Usually with SLAs in hours rather than days.
P1	High	Meaning it’s not very critical but equally important to solve quickly. In this case, we can plan the time it requires to fix, assigned resources and solve it quickly.
P2	Medium	Required to be fixed but SLAs are usually within a week.
P3	Low	Low priority defects are not at all urgent to resolve. Depending on the schedule of the development team, they can work on it to resolve the issue.

Similarly, we have different types of severities associated to the defects. Listed as below:

Code	Type	Description
S0	Critical	Defects that have a severe impact on the system like a failure of a critical functionality are marked as Critical. SLA’s are usually in hours.
S1	High	Defects found in the happy path can be classified as High severity.
S2	Medium	Medium severity defects are defects which pose issues, but it’s not that severe to be fixed immediately. Hence developers can decide the timelines to work on it. SLA’s are usually in weeks.
S3	Low	Low severity defects are again needed to be resolved, but since it’s of a low impact, it can be fixed later in the release or pushed to a future release.

To summarize, when a tester comes across an application which crashes on a happy path impacting a vast number of end users, it is marked as critical in priority and severity, i.e., P0S0 defect.

On the other hand, if we have a cosmetic defect like text on the website with underscores instead of a hyphen, then we can mark this defect with a low priority and severity, i.e., P3S3.

Defect Resolution Process

Once the defect has been discovered and assigned to a developer, the developer follows a process for root cause analysis.

A general process is to try to reproduce the defect using the details attached to the defect such as artifacts, test data, etc. The developers can reach out to the testers in case they need more information. Once the developer has all the details, they can try to find the code responsible for the functionality causing the defect. This is sometimes difficult; hence they can compare the code with a version without the defect. They can also check the logs, configuration files or even databases to see if it has any impact on the data. Debugging the application code by providing the same input values and verifying the data flows usually leads one to the root cause.

Defect Management Artifacts

Defect report and the traceability matrix are two main artifacts that get produced or updated for defect management.

Defect Report

Along with the test execution status reports, the test team also prepares defect reports. This report helps to keep all stakeholders updated and helps managers in estimation and planning the project release. It also can be used by Test Managers for resource planning and scheduling to complete the project as per deadlines.

Generally, a defect report includes the following details:

- Number of new defects logged or reopened based on test cycles
- Number of high priority and severity defects
- Number of closed defects
- Number of defects marked as duplicate, not reproducible or not a defect
- Number of defects marked as deferred
- Total number of defects created in a test cycle or release

This information is helpful to derive further conclusions such as defect density, defect rejection ratio, defect acceptance ratio, defect leakage ratio, etc. Let's discuss some of these in details.

A defect acceptance ratio shows the valid number of defects discovered during a release. It can be calculated as:

*Defect acceptance ratio (DAR) = (Number of valid defects discovered and accepted by the development team or client during the release / Total number of defects logged in the same release) * 100*

For example, let's say test team found 200 defects in a release out of which developers or client accepted 120 defects. So, the DAR ratio would be:

DAR = (120/200) *100 = 60%

Defect leakage ratio is another metric which can be described as a number of defects missed during the testing phase to the total number of defects discovered for the product.

The test manager can include as many metrics as required by the project stakeholders.

Traceability Matrix

A traceability matrix establishes a connection between the client requirements, test cases, and defects. Requirements are linked to the test cases to understand the test coverage for the specified requirement. Likewise, defects are linked to the test cases to know which test cases are failing or blocked because of new defects or existing ones which are not fixed or reopened. This map provides us with valuable insights like the number of requirements that have issues and would require more time to test while also tracking pending test cases because of the blocked, failed or untested defects. It also provides clarity for testers to put more test efforts on requirements that have been untested while it gives the developers an idea of which requirements need more attention.

One can create the requirement traceability matrix easily by adding the following details:

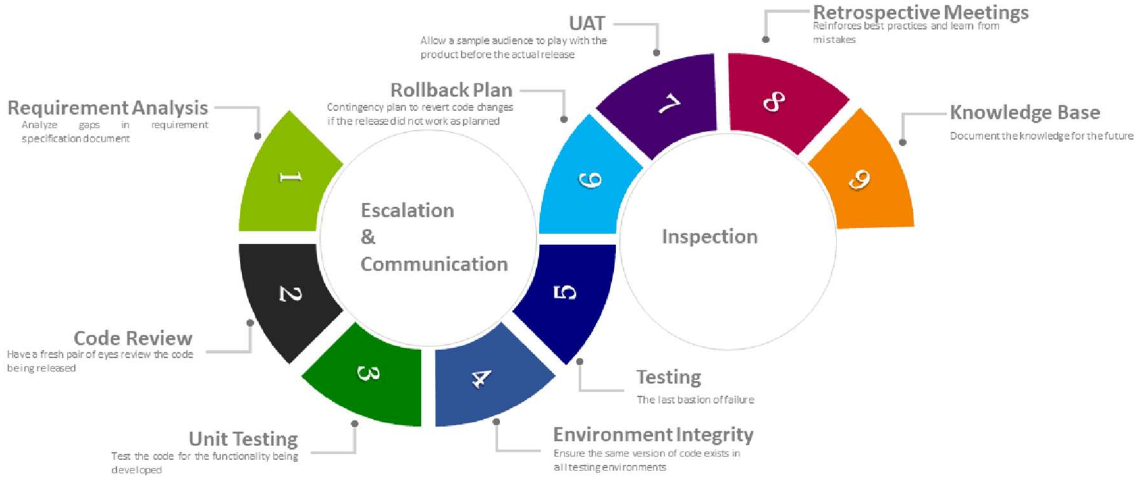
- Business Requirement ID
- Test Scenario for the business requirement
- Number of test cases based on the derived scenarios
- Test case ID
- Priority of the test cases
- Test case status
- Defect ID
- Defect status
- Comments

The structure of the traceability matrix is specific to an organization, and a company-specific template is usually available with the company PMO (Project

Management Office). The end goal, however, is to back trace the defects through the linked relevant requirements via test cases.

Steps to Prevent Defects

Fixing defects that turn up in later phases of the software development life cycle are expensive since it requires a whole lot of retrospective coding and testing which simply adds to the development time. Hence, it’s always better to identify problems and provide solutions or workarounds in the early stages of software development. This saves time and money by reducing the efforts to rework. Although defects and bugs cannot be avoided completely, here are some precautionary steps that we can take to minimize their occurrence at later stages.


Requirement Analysis
Analyze gaps in requirement specification document
Rollback Plan
Contingency plan to revert code changes if the release did not work as planned
UAT
Allow a sample audience to play with the product before the actual release
Retrospective Meetings
Reinforces best practices and learn from mistakes
Knowledge Base
Document the knowledge for the future
1
2
3
4
5
6
7
8
9
Escalation & Communication
Inspection
Code Review
Have a fresh pair of eyes review the code being released
Testing
The last bastion of failure
Unit Testing
Test the code for the functionality being developed
Environment Integrity
Ensure the same version of code exists in all testing environments

Requirement Analysis

This phase is critical in any project development. As the first and most important stage, care should be given to ensure the development and testing teams have the same understanding of the requirements. All doubts must be clarified, and any chances of dilemma should be addressed. Analyzing the gaps in the requirements can lead us to find hidden defects and fix the issues in the first stage of the development.

Code Review

Code review is another technique to reduce the number of defects passing onto later phases of the life cycle. Analyzing the data flow across various components, comparing it with the actual requirements, implementing an optimized solution and analyzing critical path scenarios can lead us to find issues at the code level. Pair programming is a good programming practice to adopt.

Unit Testing

Unit testing is performed by the developer to ensure that the code satisfies the requirements. This can be a combination of test scenarios to check input and output values, application workflows, data flows and can have checks to see if certain functions or variables return the expected values. Test Driven Development (TDD) is a good programming practice to adopt.

Environment Integrity

It is necessary to ensure that all environments for the development cycle have the same version of the application being released. Companies usually have Continuous Integration tools to keep track of deployment build and versions. Container technologies like Docker, Kubernetes and Virtual Machines can be leveraged to make this work.

Testing

Testers are the last bastions of detecting defects from being released to production. It is imperative that testers plan and validate all workflows starting from basic happy paths to the more complex ones. It’s important to discover defects in the early phases of testing life cycle. Testers must ensure that defects are re-tested and verified after being fixed.

Inspection

Inspection is an effective technique to review the test cases and identify test coverage as per requirements. It helps to identify gaps in test cases and helps improve product quality.

Rollback Plan

It is obvious that people are excited about a new product being launched. However, sometimes severe problems can be found after the production release as well. In such cases having a rollback plan can be helpful to revert to a previous working version until the issue is fixed.

Issue Escalation and Communication

Showstoppers have a serious impact on the delivery schedule of the projects hence it’s necessary to escalate such obstacles to managers so that all the stakeholders are made aware of the same.

User Acceptance Testing (UAT)

It is a type of testing where a sample of the user audience is given access to use the product with the intent to find defects. Generally, they have a list of real-world scenarios which they target on.

Identifying defects after UAT phase is useful to know if the application will be successful with a larger audience without any issues. Sometimes companies also release beta versions of the application to go through such a testing phase before launching a product globally.

Review or Retrospective Meetings

One benefit of having retrospective meetings is to assess the project performance. Listing all the issues that the team has faced and learning from them is helpful. This reinforces processes that work and evolves the team by helping them understand what doesn’t.

Knowledgebase

Creating knowledge repositories of the lessons learnt is necessary to document for future. This is helpful to train new members joining the team and can help by introducing good standards to other teams as everyone strives to find the optimum solution that works.

Conclusion

Understanding defects and managing them is thus crucial in software test life cycle. If performed correctly it aids in streamlining the project assembly line and helps the manager in project delivery. This results in delivering a quality product which meets the customer’s needs, elevates trust, and boosts business growth.