UNIT 1: Introduction to Software Testing

Syllabus: Need of testing, Basics of Software Testing, Testing Principles, Goals, Software Testing Life Cycle, Defects, Defect management, Verification and validation, Test Plan.

----

1. **Software testing:** 🎬 Software testing is defined as performing Verification and Validation of the Software Product for its correctness and accuracy of working. 🎬 Software Testing is the process of executing a program with the intent of finding errors. 🎬
A successful test is one that uncovers an as-yet-undiscovered error. 🎬 Testing can show the presence of bugs but never their absence. 🎬 Testing is a support function that helps developers look good by finding their mistakes before anyone else does.

1.1 Role of testing / Objectives of testing:

1. Finding defects which may get created by the programmer while developing the software.
2. Gaining confidence in and providing information about the level of quality.
3. To prevent defects.
4. To make sure that the end result meets the business and user requirements.
5. To ensure that it satisfies the BRS that is Business Requirement Specification and SRS that is System Requirement Specifications.
6. To gain the confidence of the customers by providing them a quality product .

1.2 What is Software testing?

• Finding defects
• Trying to break the system
• Finding and reporting defects
• Demonstrating correct functionality
• Demonstrating incorrect functionality
• Demonstrating robustness, reliability, security, maintainability, …
• Measuring performance, reliability, …
• Evaluating and measuring quality
• Proving the software correct
• Executing pre-defined test cases
• Automatic error detection

1.3 Skills Required for Tester

• Communication skills
• Domain knowledge
• Desire to learn
• Technical skills
• Analytical skills
• Planning
• Integrity
• Curiosity
• Think from users perspective
• Be a good judge of your product

1.4 What is bug, defect, error & failure give an example of each?

• A person makes an Error
• That creates a fault in software

• That can cause a failure in operation

• **Bug:** The presence of error at the time of execution of the software. A bug is the consequence/outcome of a coding fault.
• **Error:** An error is a human action that produces the incorrect result that results in a fault.
• **Fault:** State of software caused by an error.
• **Failure:** Deviation of the software from its expected result. It is an event.
• **Defect:** A defect is an error or a bug, in the application which is created. A programmer while designing and building the software can make mistakes or error. These mistakes or errors mean that there are flaws in the software. These are called defects. A Defect in Software Testing is a variation or deviation of the software application from end user's requirements or original business requirements. A software defect is an error in coding which causes incorrect or unexpected results from a software program which does not meet actual requirements. Testers might come across such defects while executing the test cases.

Why do defects occur in software?

Software is written by human beings
Who know something, but not everything
Who have skills, but aren't perfect
Who don't usually use rigorous methods
Who do make mistakes (errors) Under increasing pressure to deliver to strict deadlines
No time to check, assumptions may be wrong
Systems may be incomplete Software is complex, abstract and invisible
Hard to understand
Hard to see if it is complete or working correctly
No one person can fully understand large systems
Numerous external interfaces and dependencies

1.5 Sources of defects

Education

Developers does not understand well enough what he or she is doing
Lack of proper education leads to errors in specification, design, coding, and testing

Communication

Developers do not know enough
Information does not reach all stakeholders
Information is lost

Oversight

Omitting to do necessary things

Transcription

Developer knows what to do but simply makes a mistake

Process

Process is not applicable for the actual situation
Process places restrictions that cause errors

Test Case A test case is a specific procedure of testing a particular requirement. It will include: • Identification of specific requirement tested • Test case success/failure criteria • Specific steps to execute test • Test Data

2.2 What are different types of software testing?

Software Testing can be broadly classified into two types:

1. **Manual Testing:** Manual testing includes testing software manually, i.e., without using any automation tool or any script. In this type, the tester takes over the role of an end-user and tests the software to identify any unexpected behavior or bug. There are different stages for manual testing such as unit testing, integration testing, system testing, and user acceptance testing.

Testers use test plans, test cases, or test scenarios to test software to ensure the completeness of testing. Manual testing also includes exploratory testing, as testers explore the software to identify errors in it.

2. **Automation Testing:** Automation testing, which is also known as Test Automation, is when the tester writes scripts and uses another software to test the product. This process involves the automation of a manual process. Automation Testing is used to re-run the test scenarios quickly and repeatedly, that were performed manually in manual testing.

Apart from regression testing, automation testing is also used to test the application from a load, performance, and stress point of view. It increases the test coverage, improves accuracy, and saves time and money when compared to manual testing.

2.2.1 Types of Software Testing Techniques

Software testing techniques can be majorly classified into two categories:

1. **Black Box Testing:** The technique of testing in which the tester doesn't have access to the source code of the software and is conducted at the software interface without any concern with the internal logical structure of the software is known as black-box testing.

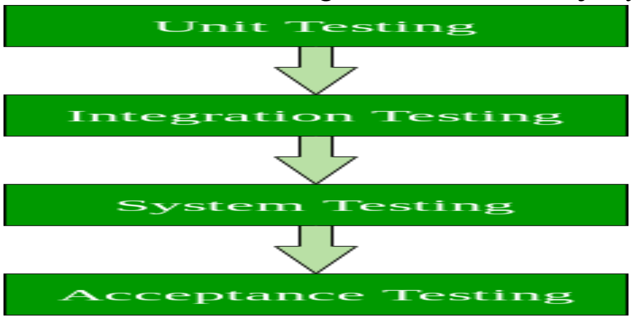
2. **White-Box Testing:** The technique of testing in which the tester is aware of the internal workings of the product, has access to its source code, and is conducted by making sure that all internal operations are performed according to the specifications is known as white box testing.

Compare Black Box and White box Testing?

Black Box Testing	White Box Testing
Internal workings of an application are not required.	Knowledge of the internal workings is a must.
Also known as closed box/data-driven testing.	Also known as clear box/structural testing.
End users, testers, and developers.	Normally done by testers and developers.
This can only be done by a trial and error method.	Data domains and internal boundaries can be better tested.

2.2.2 What are different levels of software testing?

Software level testing can be majorly classified into 4 levels:



1. **Unit Testing:** A level of the software testing process where individual units/components of a software/system are tested. The purpose is to validate that each unit of the software performs as designed.
Unit Testing is a software testing technique by means of which individual units of software i.e. group of computer program modules, usage procedures, and operating procedures are tested to determine whether they are suitable for use or not. It is a testing method using which every independent module is tested to determine if there is an issue by the developer himself. It is correlated with the functional correctness of the independent modules. Unit Testing is defined as a type of software testing where individual components of a software are tested. Unit Testing of the software product is carried out during the development of an application. An individual component may be either an individual function or a procedure. Unit Testing is typically performed by the developer. In SDLC or V Model, Unit testing is the first level of testing done before integration testing. Unit testing is such a type of testing technique that is usually performed by developers. Although due to the reluctance of developers to test, quality assurance engineers also do unit testing.
There are 3 types of Unit Testing Techniques.

Black Box Testing: This testing technique is used in covering the unit tests for input, user interface, and output parts.
White Box Testing: This technique is used in testing the functional behavior of the system by giving the input and checking the functionality output including the internal design structure and code of the modules.
Gray Box Testing: This technique is used in executing the relevant test cases, test methods, test functions, and analyzing the code performance for the modules.
Integration testing is the process of testing the interface between two software units or modules. It focuses on determining the correctness of the interface. The purpose of integration testing is to expose faults in the interaction between integrated units. Once all the modules have been unit tested, integration testing is performed.Integration testing is a software testing technique that focuses on verifying the interactions and data exchange between different components or modules of a software application. The goal of integration testing is to identify any problems or bugs that arise when different components are combined and interact with each other. Integration testing is typically performed after unit testing and before system testing. It helps to identify and resolve integration issues early in the development cycle, reducing the risk of more severe and costly problems later on.

Integration testing can be done by picking module by module. This can be done so that there should be proper sequence to be followed. And also if you don’t want to miss out on any integration scenarios then you have to follow the proper sequence. Exposing the defects is the major focus of the integration testing and the time of interaction between the integrated units.

Integration test approaches – There are four types of integration testing approaches. Those approaches are the following:

1. Big-Bang Integration Testing – It is the simplest integration testing approach, where all the modules are combined and the functionality is verified after the completion of individual module testing. In simple words, all the modules of the system are simply put together and tested. This approach is practicable only for very small systems. If an error is found during the integration testing, it is very difficult to localize the error as the error may potentially belong to any of the modules being integrated. So, debugging errors reported during big bang integration testing is very expensive to fix.

2. Bottom-Up Integration Testing – In bottom-up testing, each module at lower levels is tested with higher modules until all modules are tested. The primary purpose of this integration testing is that each subsystem tests the interfaces among various modules making up the subsystem. This integration testing uses test drivers to drive and pass appropriate data to the lower level modules.

3. Top-Down Integration Testing – Top-down integration testing technique is used in order to simulate the behaviour of the lower-level modules that are not yet integrated. In this integration testing, testing takes place from top to bottom. First, high-level modules are tested and then low-level modules and finally integrating the low-level modules to a high level to ensure the system is working as intended.

4. Mixed Integration Testing – A mixed integration testing is also called sandwiched integration testing. A mixed integration testing follows a combination of top down and bottom-up testing approaches. In top-down approach, testing can start only after the top-level module have been coded and unit tested. In bottom-up approach, testing can start only after the bottom level modules are ready. This sandwich or mixed approach overcomes this shortcoming of the top-down and bottom-up approaches. It is also called the hybrid integration testing. also, stubs and drivers are used in mixed integration testing.

2. **System Testing:** A level of the software testing process where a complete, integrated system/software is tested. The purpose of this test is to evaluate the system's compliance with the specified requirements. **System Testing** is a type of software testing that is performed on a complete integrated system to evaluate the compliance of the system with the corresponding requirements. In system testing, integration testing passed components are taken as input. The goal of integration testing is to detect any irregularity between the units that are integrated together. System testing detects defects within both the integrated units and the whole system. The result of system testing is the observed behavior of a component or a system when it is tested. **System Testing** is carried out on the whole system in the context of either system requirement specifications or functional requirement specifications or in the context of both. System testing tests the design and behavior of the system and also the expectations of the customer. It is performed to test the system beyond the bounds mentioned in the software requirements specification (SRS). System Testing is basically performed by a testing team that is independent of the development team that helps to test the quality of the system impartial. It has both functional and

non-functional testing. **System Testing is a black-box testing**. System Testing is performed after the integration testing and before the acceptance testing.

5.2 **Types of System Testing:**

Performance Testing: Performance Testing is a type of software testing that is carried out to test the speed, scalability, stability and reliability of the software product or application.

Load Testing: Load Testing is a type of software Testing which is carried out to determine the behavior of a system or software product under extreme load.

Stress Testing: Stress Testing is a type of software testing performed to check the robustness of the system under the varying loads.

Scalability Testing: Scalability Testing is a type of software testing which is carried out to check the performance of a software application or system in terms of its capability to scale up or scale down the number of user request load.

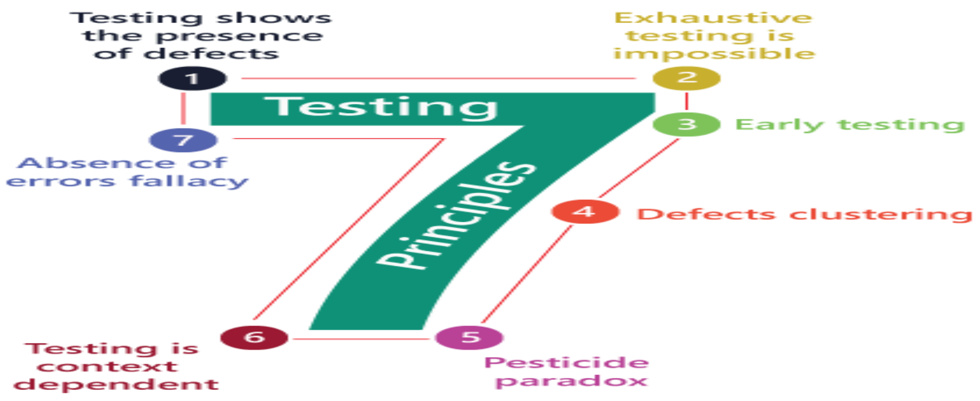
1. **Acceptance Testing:** A level of the software testing process where a system is tested for acceptability. The purpose of this test is to evaluate the system's compliance with the business requirements and assess whether it is acceptable for delivery or Not.

3.Explain Software Testing Principles?

Software testing is a procedure of implementing software or the application to identify the defects or bugs. For testing an application or software, we need to follow some principles to make our product defects free, and that also helps the test engineers to test the software with their effort and time. Here, in this section, we are going to learn about the seven essential principles of software testing.

Let us see the seven different testing principles, one by one:

- Testing shows the presence of defects
- Exhaustive Testing is not possible
- Early Testing
- Defect Clustering
- Pesticide Paradox
- Testing is context-dependent
- Absence of errors fallacy


Testing shows the presence of defects
1
Exhaustive testing is impossible
2
Testing
3
Early testing
7
Absence of errors fallacy
Principles
4
Defects clustering
6
Testing is context dependent
5
Pesticide paradox

Testing shows the presence of defects

The test engineer will test the application to make sure that the application is bug or defects free. While doing testing, we can only identify that the application or software has any errors. The primary purpose of doing testing is to identify the numbers of unknown bugs with the help of various methods and testing techniques because the entire test should be traceable to the customer requirement, which means that to find any defects that might cause the product failure to meet the client's needs.

By doing testing on any application, we can decrease the number of bugs, which does not mean that the application is defect-free because sometimes the software seems to be bug-free while performing multiple types of testing on it. But at the time of deployment in the production server, if the end-user encounters those bugs which are not found in the testing process.

Exhaustive Testing is not possible

Sometimes it seems to be very hard to test all the modules and their features with effective and non- effective combinations of the inputs data throughout the actual testing process.

Hence, instead of performing the exhaustive testing as it takes boundless determinations and most of the hard work is unsuccessful. So we can complete this type of variations according to the importance of the modules because the product timelines will not permit us to perform such type of testing scenarios.

Early Testing

Here early testing means that all the testing activities should start in the early stages of the software development life cycle's **requirement analysis stage** to identify the defects because if we find the bugs at an early stage, it will be fixed in the initial stage itself, which may cost us very less as compared to those which are identified in the future phase of the testing process.

To perform testing, we will require the requirement specification documents; therefore, if the requirements are defined incorrectly, then it can be fixed directly rather than fixing them in another stage, which could be the development phase.

Defect clustering

The defect clustering defined that throughout the testing process, we can detect the numbers of bugs which are correlated to a small number of modules. We have various reasons for this, such as the modules could be complicated; the coding part may be complex, and so on.

These types of software or the application will follow the **Pareto Principle**, which states that we can identify that approx. Eighty percent of the complication is present in 20 percent of the modules. With the help of this, we can find the uncertain modules, but this method has its difficulties if the same tests are performing regularly, hence the same test will not able to identify the new defects.

Pesticide paradox

This principle defined that if we are executing the same set of test cases again and again over a particular time, then these kinds of the test will not be able to find the new bugs in the software or the application. To get over these pesticide paradoxes, it is very significant to review all the test cases frequently. And the new and different tests are necessary to be written for the implementation of multiple parts of the application or the software, which helps us to find more bugs.

Testing is context-dependent

Testing is a context-dependent principle states that we have multiple fields such as e-commerce websites, commercial websites, and so on are available in the market. There is a definite way to test the commercial site as well as the e-commerce websites because every application has its own needs, features, and functionality. To check this type of application, we will take the help of various kinds of testing, different technique, approaches, and multiple methods. Therefore, the testing depends on the context of the application.

Absence of errors fallacy

Once the application is completely tested and there are no bugs identified before the release, so we can say that the application is 99 percent bug-free. But there is the chance when the application is tested beside the incorrect requirements, identified the flaws, and fixed them on a given period would not help as testing is done on the wrong specification, which does not apply to the client's requirements. The absence of error fallacy means identifying and fixing the bugs would not help if the application is impractical and not able to accomplish the client's requirements and needs.

4. Explain Software Testing Goal?

The main goal of software testing is to find bugs as early as possible and fix bugs and make sure that the software is bug-free. The goals of software testing may be classified into three major categories as follows:

1. **Immediate Goals**
2. **Long-term Goals**
3. **Post-Implementation Goals**

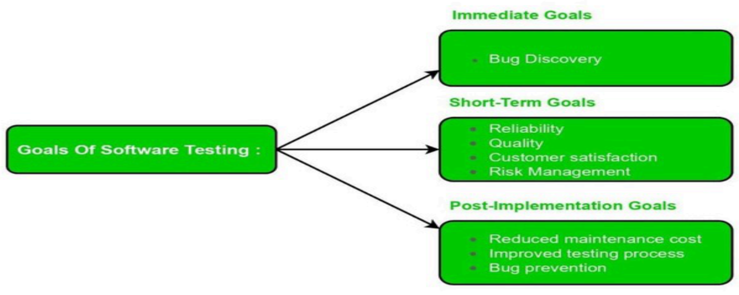


Fig : Software Testing Goals

1. Immediate Goals: These objectives are the direct outcomes of testing. These objectives may be set at any time during the SDLC process. Some of these are covered in detail below:

- **Bug Discovery:** This is the immediate goal of software testing to find errors at any stage of software development. The number of bugs is discovered in the early stage of testing. The primary purpose of software testing is to detect flaws at any step of the development process. The higher the number of issues detected at an early stage, the higher the software testing success rate.

- **Bug Prevention:** This is the immediate action of bug discovery, that occurs as a result of bug discovery. Everyone in the software development team learns how to code from the behavior and analysis of issues detected, ensuring that bugs are not duplicated in subsequent phases or future projects.

2. Long-Term Goals: These objectives have an impact on product quality in the long run after one cycle of the SDLC is completed. Some of these are covered in detail below:

- **Quality:** This goal enhances the quality of the software product. Because software is also a product, the user's priority is its quality. Superior quality is ensured by thorough testing. Correctness, integrity, efficiency, and reliability are all aspects that influence quality. To attain quality, you must achieve all of the above-mentioned quality characteristics.
- **Customer Satisfaction:** This goal verifies the customer's satisfaction with a developed software product. The primary purpose of software testing, from the user's standpoint, is customer satisfaction. Testing should be extensive and thorough if we want the client and customer to be happy with the software product.
- **Reliability:** It is a matter of confidence that the software will not fail. In short, reliability means gaining the confidence of the customers by providing them with a quality product.
- **Risk Management:** Risk is the probability of occurrence of uncertain events in the organization and the potential loss that could result in negative consequences. Risk management must be done to reduce the failure of the product and to manage risk in different situations.

3. Post Implemented Goals: After the product is released, these objectives become critical. Some of these are covered in detail below:

- **Reduce Maintenance Cost:** Post-released errors are costlier to fix and difficult to identify. Because effective software does not wear out, the maintenance cost of any software product is not the same as the physical cost. The failure of a software product due to faults is the only expense of maintenance. Because they are difficult to discover, post-release mistakes always cost more to rectify. As a result, if testing is done thoroughly and effectively, the risk of failure is lowered, and maintenance costs are reduced as a result.
- **Improved Software Testing Process:** These goals improve the testing process for future use or software projects. These goals are known as post-implementation goals. A project's testing procedure may not be completely successful, and there may be room for improvement. As a result, the bug history and post-implementation results can be evaluated to identify stumbling blocks in the current testing process that can be avoided in future projects.

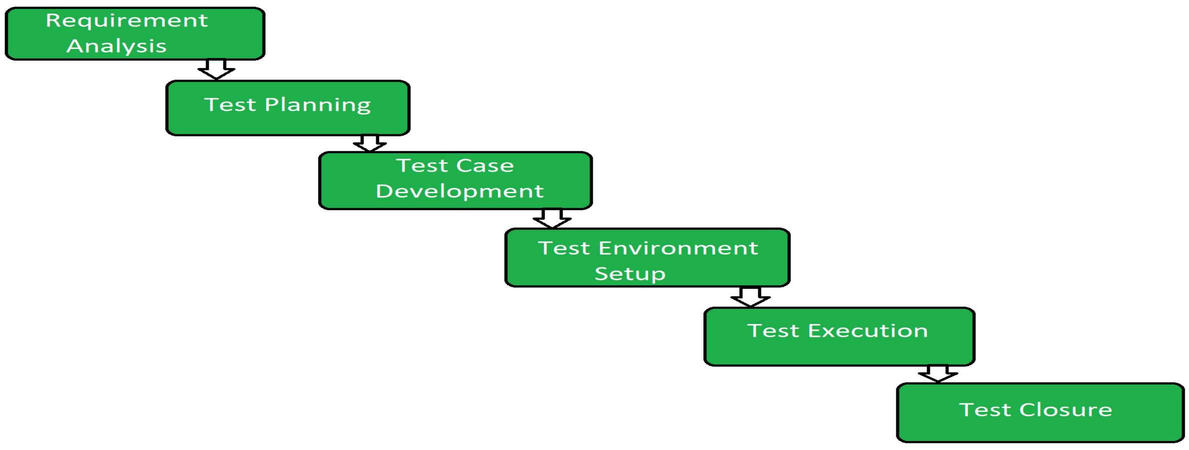
Expalain Software Testing Life Cycle (STLC)?

Software Testing Life Cycle (STLC) is a sequence of different activities performed during the software testing process.

Characteristics of STLC:

STLC is a fundamental part of Software Development Life Cycle (SDLC) but STLC consists of only the testing phases.
STLC starts as soon as requirements are defined or software requirement document is shared by stakeholders.
STLC yields a step-by-step process to ensure quality software.
In the initial stages of STLC, while the software product or the application is being developed, the testing team analyzes and defines the scope of testing, entry and exit criteria and also the test cases. It helps to reduce the test cycle time and also enhance the product quality.
As soon as the development phase is over, testing team is ready with test cases and start the execution. This helps in finding bugs in the early phase.

5.Phases of STLC:



1. **Requirement Analysis:**
   Requirement Analysis is the first step of Software Testing Life Cycle (STLC). In this phase quality assurance team understands the requirements like what is to be tested. If anything is missing or not understandable then quality assurance team meets with the stakeholders to better understand the detail knowledge of requirement.
2. **Test Planning:**
   Test Planning is most efficient phase of software testing life cycle where all testing plans are defined. In this phase manager of the testing team calculates estimated effort and cost for the testing work. This phase gets started once the requirement gathering phase is completed.
3. **Test Case Development:**
   The test case development phase gets started once the test planning phase is completed. In this phase testing team note down the detailed test cases. Testing team also prepare the required test data for the testing. When the test cases are prepared then they are reviewed by quality assurance team.
4. **Test Environment Setup:**
   Test environment setup is the vital part of the STLC. Basically test environment decides the conditions on which software is tested. This is independent activity and

can be started along with test case development. In this process the testing team is not involved. either the developer or the customer creates the testing environment.

5. **Test Execution:**
   After the test case development and test environment setup test execution phase gets started. In this phase testing team start executing test cases based on prepared test cases in the earlier step.
6. **Test Closure:**
   This is the last stage of STLC in which the process of testing is analyzed.

7. Defect in Software Testing

- The bug announced by the **programmer** and inside the code is called a
- In other words, we can say that when the application is not working as per the requirement is knows as **defects**.
- It is specified as the irregularity from the **actual and expected result** of the application or software.
- The **Defect** is the difference between the actual outcomes and expected outputs.
- The **Test engineer** can identify the defect, and it was fixed by the developer in the development phase of the **software development life cycle**.
- When a test engineer tests a piece of code, he/she comes across differences in expected output to the existing output, which is known as a **defect**. And the substitute of defect can be further known as **issues, bugs, and incidents** in software testing.

What is Defect Management Process?

The **defect management process** is the core of software testing. Once the defects have been identified, the most significant activity for any organization is to manage the flaws, not only for the testing team but also for everyone involved in the software development or project management process.
As we know, **defect prevention** is an effective and efficient way to decrease the number of defects. The defect prevention is a very cost-effective process to fix those defects discovered in the earlier stages of software processes.
The **Defect Management Process** is process where most of the organizations manage the **Defect Discovery, Defect Removal**, and then the **Process Improvement**.
As the name recommends**, the Defect Management Process (DMP)** manages defects by purely detecting and resolving or fixing the faults.
It is impossible to make a software 100% error or defect-free, but several defects can be declined by fixing or resolving them.
The defect management process primarily focuses on stopping defects, finding defects in the earlier stages, and moderating the effect of defects.

The Objective of Defect Management Process (DMP)

The main objective of the defect management process is as discussed below:

The primary objective of DMP is to expose the defects at an early stage of the software development process.
The execution of the defect management process will help us enhance the process and implementation of software.
The defect management process reduces the impact or effects of defects on software.
The Defect management process (DMP) helps us to avoid defects.
The main goal of the Defect management process is to resolve or fixing the defects.

And for the **different organization or projects** the critical goals of the Defect management process is as follows:

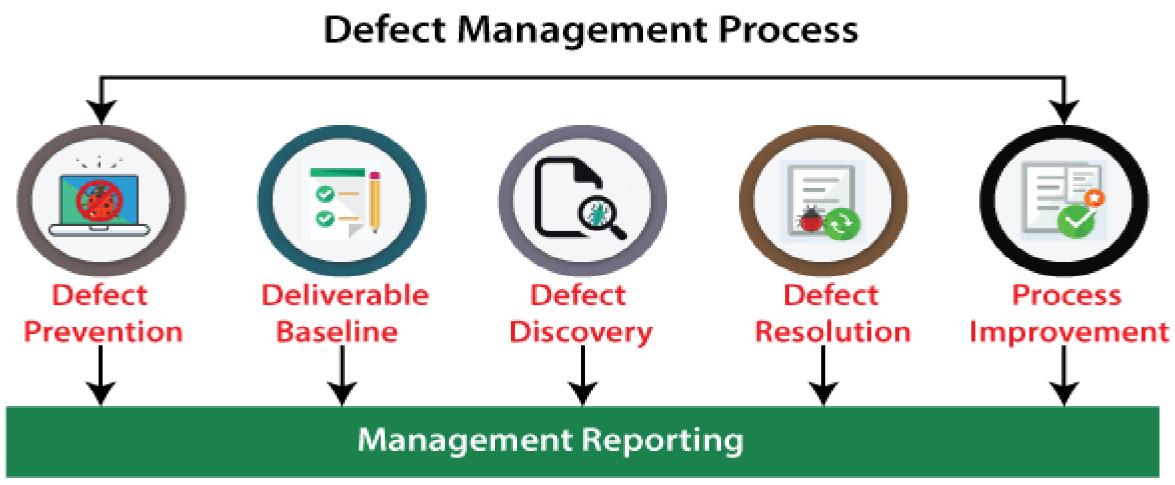
The defect management process allows us to provide input for status and progress reports about the defect.
To find the primary cause that how the defect happened and how to handle it.
To provide input, for information related to the release of the defect.

Various Stages of Defect Management Process

The defect management process includes several stages, which are as follows:

1. **Defect Prevention**
2. **Deliverable Baseline**
3. **Defect Discovery**
4. **Defect Resolution**
5. **Process Improvement**
6. **Management Reporting**

Let's discuss them one by one:

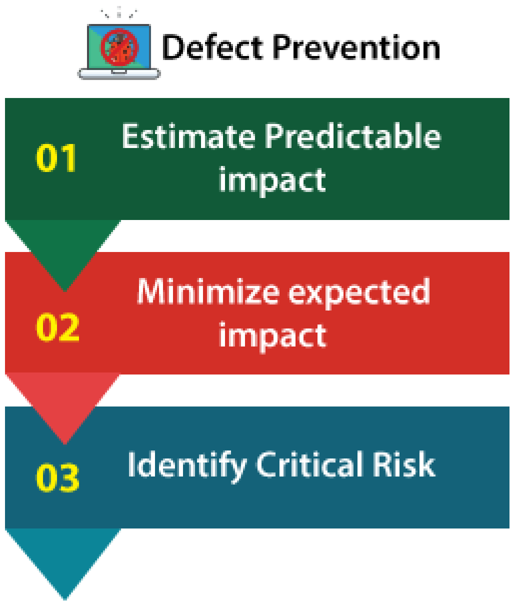


1. Defect Prevention

The first stage of the **defect management process** is **defect prevention**. In this stage, the execution of procedures, methodology, and standard approaches decreases the risk

of defects. Defect removal at the initial phase is the best approach in order to reduction its impact.
Because in the initial phase of fixing or resolving defects is less expensive, and the impact can also be diminished.
But for the future phases, identifying faults and then fixing it is an expensive process, and the effects of defect can also be amplified.
The defect prevention stage includes the following significant steps:

- **Estimate Predictable Impact**
- **Minimize expected impact**
- **Identify Critical Risk**



Step1: Estimate Predictable Impact
In this step, if the risk is encountered, then we can calculate the estimated financial impact for every critical occasion.
Step2: Minimize expected impact
When all the critical risk has been discovered, we can take the topmost risks that may be dangerous to the system if encountered and try to diminish or eliminate it.
Those risks that cannot be removed will decrease the possibility of existence and its financial impact.
Step3: Identify Critical Risk
In defect prevention, we can quickly identify the system's critical risks that will affect more if they happened throughout the testing or in the future stage.

2. Deliverable Baseline

The second stage of the defect management process is the **Deliverable baseline.** Here, the deliverable defines the **system, documents, or product**.
We can say that the **deliverable is a baseline** as soon as a deliverable reaches its pre-defined milestone.
In this stage, the deliverable is carried from one step to another; the system's existing defects also move forward to the next step or a milestone.

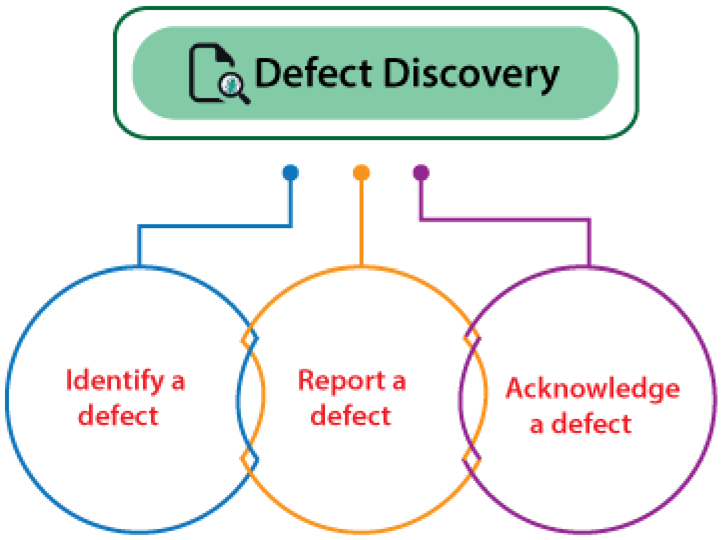
In other words, we can say that once a deliverable is baselined, any additional changes are controlled.

3. Defect Discovery

The next stage of the defect management process is **defect discovery**. At the early stage of the defect management process, defect discovery is very significant. And later, it might cause greater damage.

If developers have approved or documented the defect as a valid one, then only a defect is considered as **discovered**.
As we understood that, it is practically impossible to eliminate each defect from the system and make a system defect-free. But we can detect the defects early before they become expensive to the project.
The following phases have been including in the defect discovery stage; let's understand them in details:

- **Identify a defect**
- **Report a defect**
- **Acknowledge Defect**



Phase1: Identify a Defect
In the first phase of defect discovery where we need to find the defects before becoming a critical problem.

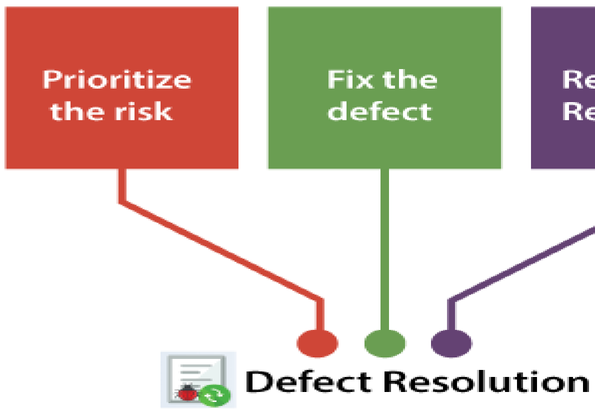
Phase2: Report a Defect

The moment testing team identifies a defect, they need to assign known issues to the development team for further evaluation and fixing process.
Phase3: Acknowledge Defect
Once the test engineers' hand over the defect to the assigned developers, now it is the responsibility of development teams to acknowledge the fault and remain further to fix it if the defect is a valid one.

4. Defect Resolution

Once the **defect discovery** stage has been completed successfully, we move to the next step of the defect management process, **Defect Resolution**.
The **Defect Resolution** is a step-by-step procedure of fixing the defects, or we can say that this process is beneficial in order to specified and track the defects.
This process begins with handing over the defects to the development team. The developers need to proceed with the resolution of the defect and fixed them based on the **priority.**
Once the defect has been selected, the developer sends a defect report of resolution to the test manager's testing team.
The defect resolution process also involves the notification back to the test engineer to confirm that the resolution is verified.
We need to follow the below steps in order to accomplish the defect resolution stage.

- **Prioritize the risk**
- **Fix the defect**
- **Report the Resolution**



Step1: Prioritize the risk
In the first step of defect resolution, the development team evaluates the defects and arranges the fault's fixing. If a defect is more impactful on the system, then developers need to fix those defects on a high priority.
Step2: Fix the defect
In the second step, the developer will fix the defects as per the priority, which implies that the higher priority defects are resolved first. Then the developer will fix the lower priority defects.
Step3: Report the Resolution
In the last step of defect resolution, the developer needs to send the fixed defects report. As it is the responsibility of development teams to make sure that the testing team is well aware of when the defects are going to be fixed and how the fault has been fixed.
This step will be beneficial for the testing team's perspective to understand the root of the defect.

5. Process Improvement

In the above stage (defect resolution), the defects have been arranged and fixed.
Now, in the **process improvement** phase, we will look into the lower priority defects because these defects are also essential as well as impact the system.

All the acknowledged defects are equal to a critical defect from the process improvement phase perspective and need to be fixed.
The people involved in this particular stage need to recall and check from where the defect was initiated.
Depending on that, we can make modifications in the **validation process, base-lining document, review process** that may find the flaws early in the process, and make the process less costly.
These minor defects allow us to learn how we can enhance the process and avoid the existence of any kind of defects that may affect the system or the product failure in the future.

6. Management Reporting

Management reporting is the last stage of the **defect management process**. It is a significant and essential part of the defect management process. The management reporting is required to make sure that the generated reports have an objective and increase the defect management process.
In simple words, we can say that the evaluation and reporting of defect information support organization and risk management, process improvement, and project management.
The information collected on specific defects by the project teams is the root of the management reporting. Therefore, every organization needs to consider the information gathered throughout the defect management process and the grouping of individual defects.

Defect Workflow and States

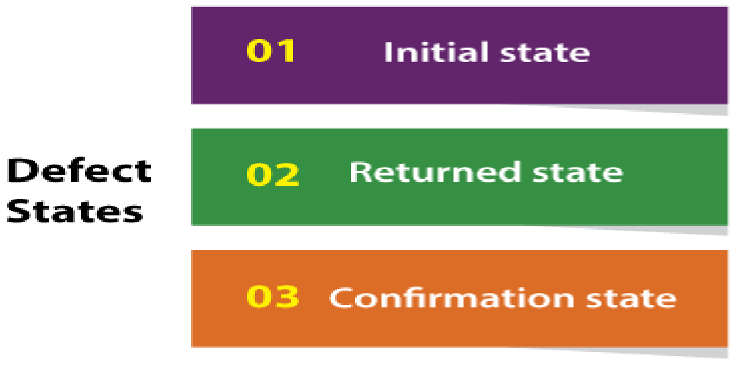
Various organizations that achieve the **software testing** with the help of a tool that keeps track of the defects during the **bug/defect lifecycle** and also contains defect reports.
Generally, one owner of the defects reports at each state of **defect lifecycle**, responsible for finishing a task that would move defect report to the successive state.
Sometimes, **defect report may not have an owner** in the last phases of the defect lifecycle if we may face the following situation:

- If the defect is invalid, then the Defect report is **cancelled**.
- The defect report is considered **deferred** if the defect won't be fixed as part of the project.
- If the fault cannot be detected anymore, hence defect report is regarded as not **reproducible**.
- The defect report is considered **closed** if the defect has been fixed and tested.

Defect States

If defects are identified throughout the testing, the testing team must manage them in the following three states:

- **Initial state**
- **Returned state**
- **Confirmation state**



1. Initial State

- It is the first state of defect, which is also known as open state.
- One or several test engineers are responsible for collecting all required data to fix the defects in this state.

2. Returned state

- The second state of defect is **returned state**. In this, the person receiving the test report rejects and asks the report creator to provide further information.
- In a returned state, the test engineers can provide more information or accept the rejection of the report.
- If various reports are rejected, the test manager should look out for faults in the initial information collection process itself.
- The returned state is also referred as the **clarification state or rejected state**.

3. Confirmation state

- The last state of defect is **the confirmation state**, where the test engineer performed a **confirmation testing**to make sure that the defect has been fixed.
- It is achieved by repeating the steps, which found the defect at the time of testing.
- If the defect is resolved, then the report is closed.
- And if the defect was not resolved, then the report is considered as **re-opened** and reported back to the owner who formerly preserved the defect report for fixing.
- A confirmation state is also known as **a verified or resolved state**.

Advantages of Defect Management Process

Following are the most significant benefits of the Defect management process:

Confirm Resolution

The defect management process will also help us to make sure the resolution of defects being tracked.

Accessibility of Automation Tools

One of the most significant procedures of the defect management process is the **defect or bug tracking process**.

For defect tracking, we have various automation tools available in the market, which can help us to track the defect in the early stages.

These days, various different tools are available in order to track different types of defects. **For example,**

Software Tools: These types of tools are used to identified or track non-technical problems.

User-facing Tools: These types of tools will help us to discover the defects, which are related to production.

Offer Valuable Metrics

The defect management process is also offering us valuable defect metrics together with automation tools.

And these valuable defect metrics help us in reporting and continuous enhancements.

Disadvantages of Defect Management Process

The drawbacks of the defect management process are as follows:

- If the defect management process is not performed appropriately, then we may have a loss of customers, loss of revenue, and damaged brand reputations.
- If the defect management process is not handled properly, then there will a huge amplified cost in a creeping that is a rise in the price of the product.
- If defects are not accomplished appropriately at an early stage, then afterward, the defect might cause greater damage, and costs to fix the defects will also get enhanced.

Verification and Validation

Verification and Validation is the process of investigating that a software system satisfies specifications and standards and it fulfills the required purpose. **Barry Boehm** described verification and validation as the following:

Verification: *Are we building the product right?*

Validation: *Are we building the right product?*

Verification:

Verification is the process of checking that a software achieves its goal without any bugs. It is the process to ensure whether the product that is developed is right or not. It verifies whether the developed product fulfills the requirements that we have.

Verification is **Static Testing**.

Activities involved in verification:

1. Inspections
2. Reviews
3. Walkthroughs

4. Desk-checking

Validation:

Validation is the process of checking whether the software product is up to the mark or in other words product has high level requirements. It is the process of checking the validation of product i.e. it checks what we are developing is the right product. it is validation of actual and expected product.

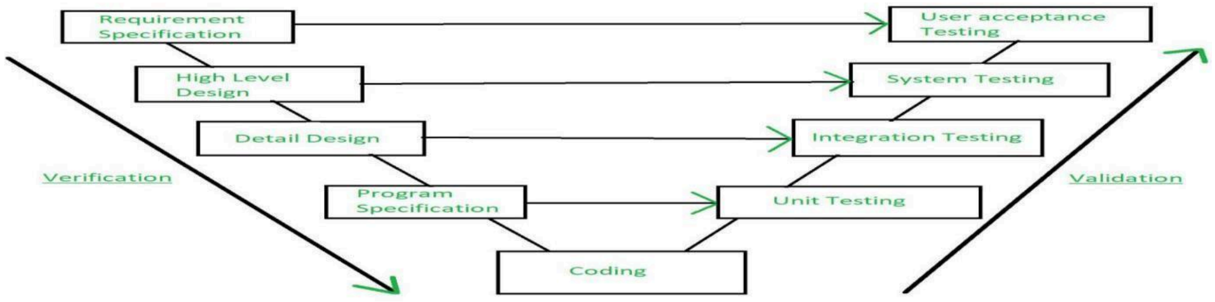
Validation is the **Dynamic Testing**.

Compare Verification and Validation

Verification	**Validation**
The verifying process includes checking documents, design, code, and program	It is a dynamic mechanism of testing and validating the actual product
It does ***not*** involve executing the code	It always involves executing the code
Verification uses methods like reviews, walkthroughs, inspections, and desk- checking etc.	It uses methods like Black Box Testing, White Box Testing, and non-functional testing
Whether the software conforms to specification is checked	It checks whether the software meets the requirements and expectations of a customer
It finds bugs early in the development cycle	It can find bugs that the verification process can not catch
Target is application and software architecture, specification, complete design, high level, and database design etc.	Target is an actual product
QA team does verification and make sure that the software is as per the requirement in the SRS document.	With the involvement of testing team validation is executed on software code.
It comes before validation	It comes after verification

Activities involved in validation:

1. Black box testing
2. White box testing
3. Unit testing
4. Integration testing



7. Test Plan

A test plan is a systematic approach to testing a system i.e. software. The plan typically contains a detailed understanding of what the eventual testing workflow will be. A test plan is a document that consists of all future testing-related activities. It is prepared at the project level and in general, it defines work products to be tested, how they will be tested, and test type distribution among the testers. Before starting testing there will be a test manager who will be preparing a test plan. In any company whenever a new project is taken up before the tester involves in the testing the test manager of the team would prepare a test Plan.

7.1 Importance of Test Plan

The following are some of the key benefits of making a test plan:

- It acts as a quick guide for the testing process.
- It helps to avoid out-of-scope functionalities.
- It determines the time, cost, and effort.
- Provide a schedule for testing activities.
- Resource requirement and equipment.
- Test Plan Document can be used for similar projects.
- It helps to understand the test details.
- It helps in determining the quality of software applications.

7.2 Test Plan Guidelines

- Avoid Overlapping and repetition.
- Avoid Lengthy Paragraph.
- Use lists and tables.
- Update plan.
- Don’t use outdated documents.

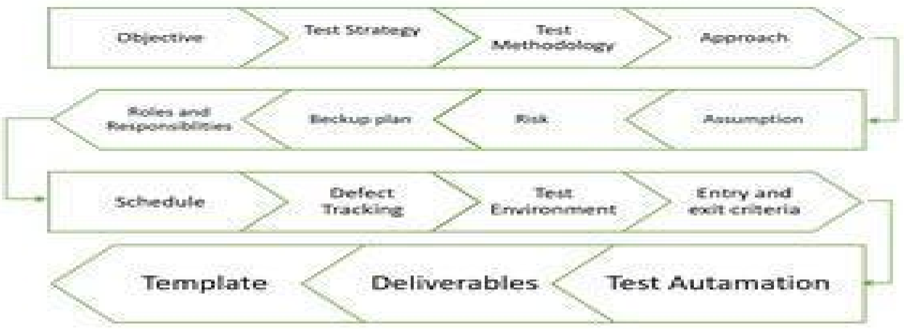
7.3 Type of Test Plan

The following are the three types of test plans:

- **Master Test Plan-** In this type of test plan, includes multiple test strategies and has multiple levels of testing.
- **Phase Test Plan-** In this type of test plan, emphasis on any one phase of testing.
- **Specific Test Plan-** In this type of test plan, it is designed for specific types of testing especially non-functional testing.

7.4 ExplainTest Plan Attributes?

There is no hard and fast rule of preparing a test plan but it has some **standard 15 attributes** that companies follow:



A. Objective: It describes the aim of the test plan, whatever the good process and procedure they are going to follow in order to give quality software to customers. The overall objective of the test is to find as many defects as possible and to make software bug free. The test objective must be broken into components and sub-components. In every component following activities should be performed.

- List all the functionality, performance to be tested.
- Make goals and targets based on the application feature.

B. Test Strategy: It is a crucial document that is to be performed and usually designed by the Test Manager. It helps to determine Test Effort and Test cost. Test strategy helps to determine the features that are going to be tested and the features that will not be tested. The scope can be divided into two parts:

- **In-Scope:** The modules that are to be tested rigorously.
- **Out Scope:** The modules that are not to be tested rigorously.

Example: In an application A, B, C, D features have to be developed, but the B feature has already been designed by other companies. So the development team will purchase B from that company and perform only integrated testing with A, B, C.

C. Testing Methodology: The methods that are going to be used for testing depend on application to application. The testing methodology is decided based on the feature and application requirements.

Since the testing terms are not standard, one should define what kind of testing will be used in the testing methodology. So that everyone can understand it.

D. Approach: The approach of testing different software is different. It deals with the flow of applications for future references. It has two aspects:

High-Level Scenarios: For testing critical features high-level scenarios are written. For Example, login to a website, booking from a website.
The Flow Graph: It is used when one wants to make benefits such as converging and merging easy.
E. Assumptions: In this phase, certain assumptions will be made.

Example:

The testing team will get proper support from the development team.
The tester will get proper knowledge transfer from the development team.
Proper resource allocation will be given by the company to the testing department.
F. Risk: All the risks that can happen if the assumption is breaking. For Example, in the case of wrong budget estimation, the cost may overrun. Some reason that may lead to risk is:

Test Manager has poor management skills.
Hard to complete the project on time.
Lack of cooperation.
G. Backup/Mitigation Plan- If any risk is involved then the company must have a backup plan, the purpose is to avoid errors. Some points to resolve/avoid risk:

Test priority is to be set for each test activity.
Managers should have leadership skills.
Training course for the testers.
H. Roles and Responsibilities: All the responsibilities and role of every member in a particular testing team has to be recorded.

Example:

Test Manager: Manages the project, takes an appropriate resource and gives project direction.
Tester: Identify the testing technique, verify the test approach, and save project cost.
I. Scheduling: Under this, it will record the start and the end date of each and every testing-related activity. For Example, writing test case date and ending test case date.

J. Defect Tracking: It is an important process in software engineering as lots of issue arises when you develop a critical system for business. If there is any defect found while testing and that defect must be given to the developer team. There are the following methods for the process of defect tracking:

K. Information Capture: In this, we take basic information to begin the process.
Prioritize: The task is prioritized based on severity and importance.
Communicate: Communication between the identifier of bug and fixer of bug.
Environment: Test the application based on hardware and software.
Example: The bug can be identified using bug tracking tools such as Jira, Mantis, Trac.

Test Environment- It is the environment which the testing team will use i.e. the list of hardware and software, while testing the application, the things which are said to be tested will be written under this section. The installation of software is also checked under this.

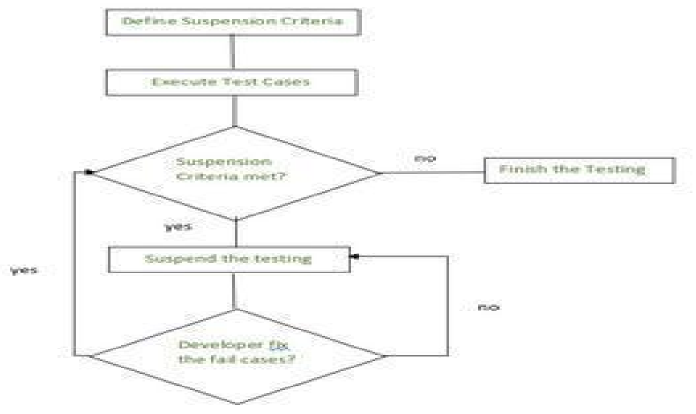
Example:

Software configuration on different operating systems, such as Windows, Linux, Mac, etc.
Hardware Configuration depends on RAM, ROM, etc.
L. Entry and Exit Criteria: The set of conditions that should be met in order to start any new type of testing or to end any kind of testing.

Entry Condition:

Necessary resources must be ready.
The application must be prepared.
Test data should be ready.
Exit Condition:

There should not be any major bug.
Most test cases should be passed.
When all test cases are executed.
Example: If the team member report 45% of the test cases failed, then testing will be suspended until the developer team fixes all defects.



Flow chart showing entry and exit condition

M.Test Automation: It consists of the features that are to be automated and which features are not to be automated.

If the feature has lots of bugs then it is categorized as Manual Testing.

If the feature is frequently tested then it can be automated.

N. Deliverables- It is the outcome from the testing team and that is to be given to the customers at the end of the project.

Before testing phase:

Test plan document.
Test case document.
Test design specification.

During testing phase:

Test scripts.
Test data.
Error logs.

After testing phase:

Test Reports.
Defect Report.
Installation Report.

It contains a test plan, defect report, automation report, assumption report, tools, and other components that have been used for developing and maintaining the testing effort.

O. Templated: It is followed by every kind of report that is going to be prepared by the testing team.

6. What If There is No Test Plan?

Below are some of the situations that may occur if there is no test plan in place:

- Misunderstanding roles and responsibilities.
- The test team will have no clear objective.
- No surety when the process ends.
- Undefined test scope may confuse testers.

Requirements Based Testing
What is Requirements Based Testing?

The process of requirements based testing deals with validating whether the requirements are complete, consistent , unambiguous, complete and logically connected. With such requirements, we can proceed to develop test cases to ensure that the test cases fullfill all the requirements. Testing in this technique revolves around requirements. The strategy of Requirement based testing is to integrate testing throughout the life cycle of the software development process, to assure quality of the Requirement Specification. The aim is defect prevention than defect detection.

Taking Requirement Based testing into account, testing is divided into the following types of activity :

- **Define Test Completion Criteria :**
  Testing should be defined in quantifiable terms. The goal is considered to be achieved only when test coverage is 100%.

- **Design Test Cases :**Test cases must be in accordance with requirements specification.
- **Build Test Cases :**Join the logical parts together to form/build test cases .
- **Execute Test Cases :**Execute the test cases to evaluate the results.

Verify Test Results : Check whether actual results deviate from the expected ones.

Verify Test Coverage : Check for functional test coverage.

Manage Test Library :

Test manager is responsible for monitoring the test case executions, that is, the tests passed or failed, or to ascertain whether all tests have been successfully performed.

Why Requirements are Critical :

Various studies have shown that software projects fail due to the following reasons:

Incomplete requirements and specifications
Frequent changes in requirements and specifications
When there is lack of user input to requirements

So the requirements based testing process addresses each of the above issues as follows :

The Requirements based testing process starts at the very early phase of the software development, as correcting issues/errors is easier at this phase.

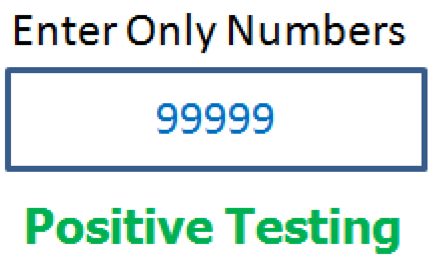
It begins at the requirements phase as the chances of occurrence of bugs have its roots here.

It aims at quality improvement of requirements. Insufficient requirements leads to failed projects.

Positive Testing

Explain Positive Testing & Negative testing with example?

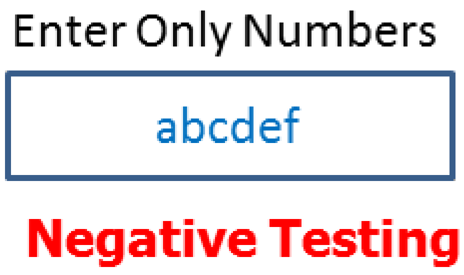
Positive Testing is a type of testing which is performed on a software application by providing the valid data sets as an input. It checks whether the software application behaves as expected with positive inputs or not. Positive testing is performed in order to check whether the software application does exactly what it is expected to do.
For example –



There is a text box in an application which can accept only numbers. Entering values up to 99999 will be acceptable by the system and any other values apart from this should not be acceptable. To do positive testing, set the valid input values from 0 to 99999 and check whether the system is accepting the values.

Negative Testing

Negative Testing is a testing method performed on the software application by providing invalid or improper data sets as input. It checks whether the software application behaves as expected with the negative or unwanted user inputs. The purpose of negative testing is to ensure that the software application does not crash and remains stable with invalid data inputs.
For example –



Negative testing can be performed by entering characters A to Z or from a to z. Either software system should not accept the values or else it should throw an error message for these invalid data inputs.

In both the testing, the following needs to be considered:

- Input data
- An action which needs to be performed
- Output Result

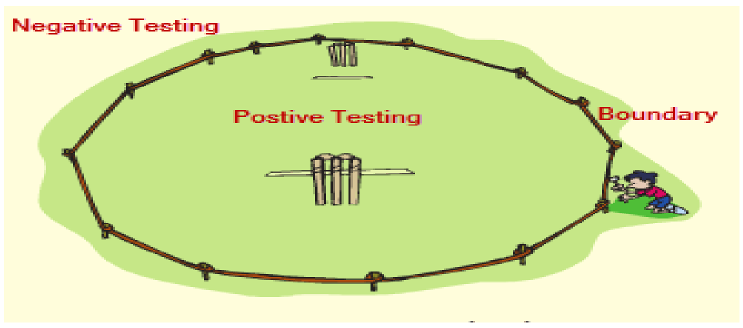
Testing Technique used for Positive and Negative Testing:

Following techniques are used for Positive and negative validation of testing is:

- Boundary Value Analysis
- Equivalence Partitioning

Boundary Value Analysis:

This is one of the software testing technique in which the test cases are designed to include values at the boundary. If the input data is used within the boundary value limits, then it is said to be Positive Testing. If the input data is picked outside the boundary value limits, then it is said to be Negative Testing.

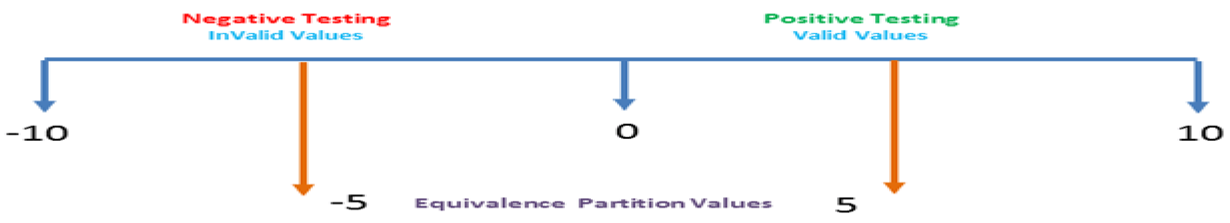


For example –

A system can accept the numbers from 0 to 10 numeric values. All other numbers are invalid values. Under this technique, boundary values -1,0,1 and 9,10,11 will be tested.

Equivalence Partitioning:

This is a software testing technique which divides the input data into many partitions. Values from each partition must be tested at least once. Partitions with valid values are used for Positive Testing. While partitions with invalid values are used for negative testing.



For example-

Numeric values Zero to ten can be divided into two (or three) partitions. In our case, we have two partitions -10 to -1 and 0 to 10. Sample values (5 and -5) can be taken from each part to test the scenarios.

UNIT 1: Introduction to Software Testing

1 . What is bug, defect, error & failure give an example of each?[5]

2. What are the skill set required by software tester?[5]

3. What are different levels of software testing? Explain in short [5]CIA 1

4. Explain the Software testing principles.[5]

5. Explain Software Testing Goal.[5]

6. Explain Software Testing Life Cycle (STLC)[5]

7. What is Defect Management Process?[5]

8. Compare Verification and Validation. [5]

9. Explain Positive Testing & Negative testing with example?[5] CIA 1

10. Compare White Box & Black Box Testing.[5]

11. What are the different errors in software testing?

12. . Explain tester's role in software development organization?

13. Explain Test Plan Attributes?

14. Explain following for testing process

A)Test strategy

B)Test planing

15. Differentiate between software testing tools and techniques.[6]

16. What is impact of defect in different phases of software development?[6]

17. Why independent testing team is required in organizations?[6]

18. What are the selection criteria of automated testing tool?

19. What is software testing process? Why do we have to test the software?

20. Define the following terms

i) Fault iii) Test bed iii) Test Case iv) Software Quality

v) Failure vi) Defect