SESSION UNDERSTANDING DOCUMENT FOR MANUAL TESTING

TABLE OF CONTENTS:

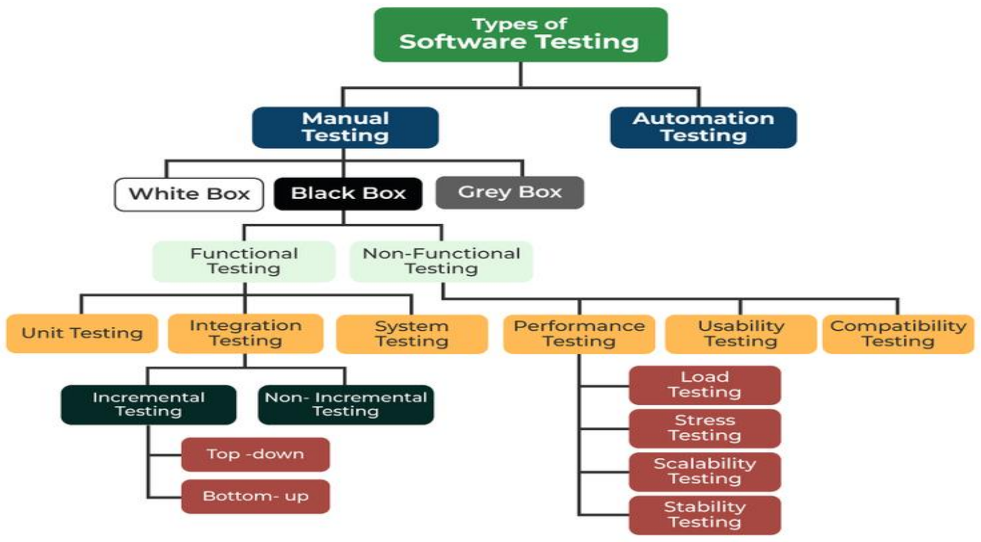
Software Testing

Software Testing is a method to check whether the actual software product matches expected requirements and to ensure that software product is defect free. The purpose of software testing is to identify errors, gaps, or missing requirements in contrast to actual requirements.

In simple terms, Software Testing means the Verification of Application Under Test (AUT).

Software Testing is Important because if there are any bugs or errors in the software, it can be identified early and can be solved before delivery of the software product. Properly tested software product ensures reliability, security, and high performance which further results in time saving, cost effectiveness and customer satisfaction.

Types of Software Testing



Functional Testing

Definition: Functional testing is a type of testing that validates the software system against the functional requirements/specifications. The purpose is to ensure that the software behaves as expected and all functionalities work correctly.

- ❖ Focuses on verifying software functions according to requirements.

Types of Functional Testing:

Unit Testing:

- Tests individual components or modules of the software.
- Usually conducted by developers.
- Tools: JUnit, NUnit, TestNG.

Integration Testing:

- Tests the interactions between integrated modules.
- Ensures combined parts work together as expected.
- Tools: Postman (API testing), JUnit (for integrated components).

System Testing:

- Tests the complete and integrated software system.
- Validates the end-to-end system specifications.
- Tools: Selenium, QTP/UFT.

User Acceptance Testing (UAT):

- Conducted by the end-users.
- Validates the software against user requirements.
- Tools: Quality Center, TestRail.

Non-Functional Testing

Definition: Non-functional testing evaluates the non-functional aspects of a software application, such as performance, usability, reliability, etc. The goal is to ensure that the software meets certain criteria related to the quality attributes.

- ❖ Focuses on evaluating the quality attributes of the software.

Types of Non-Functional Testing:

Performance Testing:

- Assesses the speed, responsiveness, and stability under a workload.
- Includes load testing, stress testing, and endurance testing.
- Tools: LoadRunner, JMeter.

Usability Testing:

- Examines how easy and user-friendly the software is.

- Focuses on the user experience and interface.
- Tools: Morae, Loop11.

Compatibility Testing:

- Verifies the software's compatibility with different environments.
- Includes operating systems, browsers, and hardware.
- Tools: BrowserStack, Sauce Labs.

7 Principles of Software Testing:

1. Testing Shows the Presence of Defects

Testing can demonstrate that defects are present, but it cannot prove that there are no defects. Testing reduces the probability of undiscovered defects remaining in the software but, even if no defects are found, it is not a proof of correctness.

The goal of testing is to find as many defects as possible to improve the software's quality.

2. Exhaustive Testing is Impossible

Testing all possible combinations of inputs and preconditions is not feasible except for trivial cases. There are too many potential paths and conditions to cover every possibility.

3. Early Testing

Testing activities should start as early as possible in the software development lifecycle. The earlier defects are detected, the cheaper they are to fix.

4. Defect Clustering

A small number of modules usually contain most of the defects discovered during pre-release testing or show the most operational failures. Focus more testing effort on the areas that are most likely to have defects.

5. Pesticide Paradox

Repeatedly running the same set of tests will not find new defects. Over time, these tests will become less effective as the software evolves. Regularly review and revise test cases, adding new and varied tests to cover different parts of the application and new features.

6. Testing is Context-Dependent

Testing is done differently in different contexts. For example, safety-critical software is tested differently from e-commerce websites.

7. Absence-of-Errors Fallacy

Finding and fixing defects does not help if the system built is unusable and does not fulfill the user's needs and expectations. Testing should ensure not only that the software is defect-free but also that it meets the business and user requirements. Validation against requirements and user acceptance testing are crucial.

Performance Testing

Definition: Performance testing is a type of non-functional testing aimed at determining the responsiveness, stability, scalability, and speed of a software application under a particular workload. The primary goal is to identify and eliminate performance bottlenecks in the software.

Objectives:

- Ensure the software application performs well under expected workloads.
- Identify and fix performance issues before the software is deployed to production.
- Verify the software's reliability and scalability.
- Provide insights into how the application behaves under stress and peak load conditions.

Types of Performance Testing

Load Testing:

Purpose: To assess how the system behaves under expected load conditions.

Description: Involves simulating a typical number of users performing a set of activities to determine if the application can handle the expected load.

Stress Testing:

Purpose: To evaluate the system's behavior under extreme load conditions.

Description: Involves pushing the system beyond its normal operational capacity to identify breaking points and performance bottlenecks.

Spike Testing:

Purpose: To determine the system's ability to handle sudden, large spikes in load.

Description: Simulates a sudden increase in the number of users or transactions to test how the system responds to abrupt changes in load.

Endurance Testing (Soak Testing):

Purpose: To verify the system's stability and performance over an extended period.

Description: Involves running the system at a normal load for an extended duration to identify potential memory leaks, performance degradation, and other issues that might arise over time.

Scalability Testing:

Purpose: To determine how well the system scales with increasing load.

Description: Involves gradually increasing the load on the system to understand its capacity and identify the maximum load it can handle before performance becomes unacceptable.

Volume Testing:

Purpose: To assess the system's ability to handle a large volume of data.

Description: Involves testing the system's performance by populating the database with a large amount of data and examining its response and behavior.

Common Performance Testing Tools

Apache JMeter: An open-source tool used for load and performance testing.

LoadRunner: A commercial performance testing tool from Micro Focus.

Gatling: An open-source tool designed for high-performance load testing.

NeoLoad: A performance and load testing tool for web and mobile applications.

BlazeMeter: A cloud-based performance testing platform compatible with JMeter.

Understanding and executing performance testing effectively ensures that software applications meet performance expectations and provide a good user experience under various conditions.

SDLC and STLC:

Software Development Life Cycle (SDLC)

Definition: The Software Development Life Cycle (SDLC) is a structured process used for developing software applications. It consists of a detailed plan describing how to develop, maintain, replace, and alter or enhance specific software. The life cycle defines a methodology for improving the quality of software and the overall development process.

Phases of SDLC:

Requirement Analysis - Gather business requirements from stakeholders and analyze them.

System Design - Define the system architecture and design based on requirements.

Implementation (Coding) - Convert the system design into source code.

Testing - Ensure the developed software is bug-free and meets the requirements.

Deployment - Deploy the application to the production environment.

Maintenance - Provide ongoing support and enhancements to the software.

Software Testing Life Cycle (STLC)

Definition: The Software Testing Life Cycle (STLC) is a sequence of specific activities conducted during the testing process to ensure software quality. Unlike SDLC, STLC focuses exclusively on the testing phases, from planning to defect tracking and closure.

Phases of STLC:

Requirement Analysis - Understand and analyze the testing requirements.

Test Planning - Define the scope and objectives of testing **(**Test strategy, resource planning, tool selection, and scheduling).

Test Case Development - Create detailed test cases and test scripts.

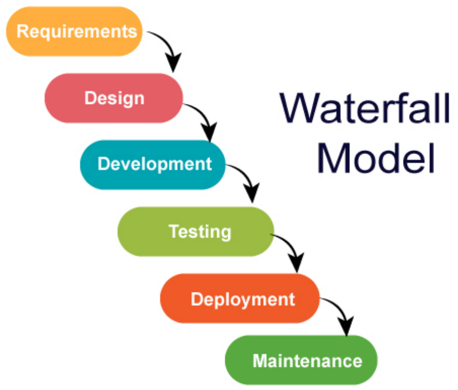
Test Environment Setup - Prepare the environment where testing will be performed.

Test Execution - Execute test cases and report defects.

Test Closure - Conclude testing activities and provide test metrics.

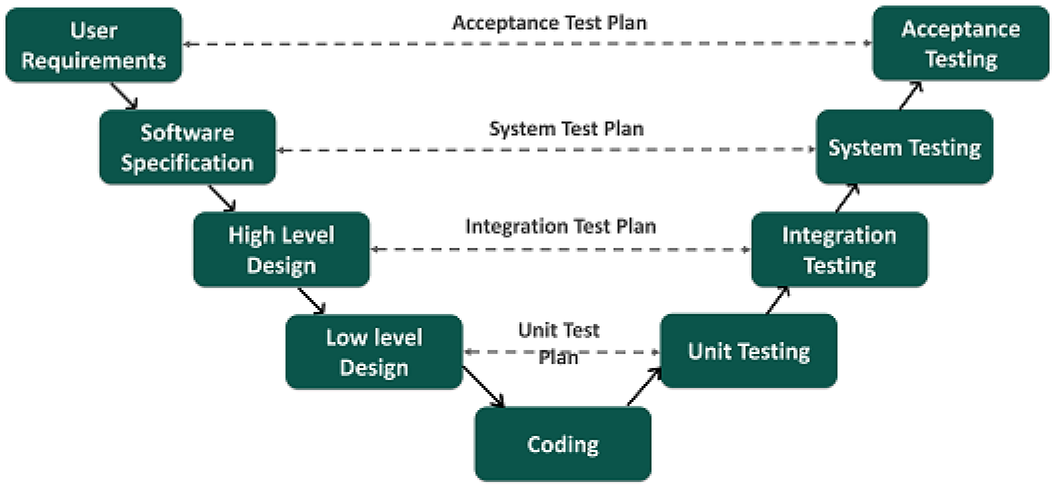
Different testing methodologies:

Waterfall Model:



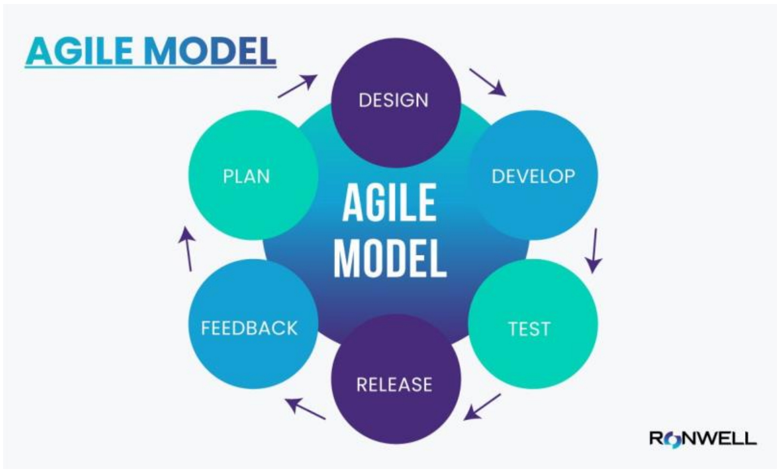
The Waterfall testing methodology is a traditional and linear approach to software development where each phase must be completed before the next one begins. This model progresses through a sequence of stages: requirement gathering and analysis, system design, implementation (coding), testing, deployment, and maintenance. Testing is a distinct phase that follows the completion of the coding phase. Each phase produces specific deliverables that feed into the next, creating a structured and disciplined process. The Waterfall model is simple and easy to understand, making it well-suited for projects with well-defined requirements. However, it is inflexible to changes and late defect detection, as testing only begins after development is complete.

V-Model:



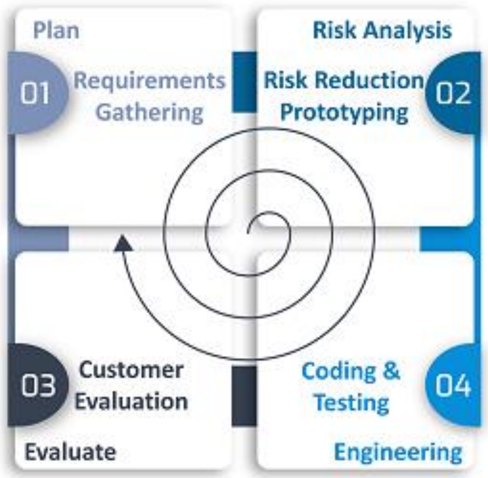
The V-Model, or Verification and Validation model, is an extension of the Waterfall methodology, emphasizing a parallel relationship between development and testing activities. For every phase in the development process, there is a corresponding testing phase. The left side of the "V" represents the development stages (requirements analysis, system design, architectural design, module design), while the right side represents the associated testing stages (system testing, acceptance testing, integration testing, unit testing). Coding sits at the bottom of the "V." This model enhances defect detection by incorporating testing activities early in the development process, ensuring that verification and validation occur throughout the lifecycle. However, like the Waterfall model, it can be rigid and assumes that requirements are clear from the start.

Agile Model:



Agile testing is a dynamic and iterative approach that is integral to Agile software development. Unlike traditional methodologies, Agile testing does not wait for development to complete before starting. Instead, testing is continuous and concurrent with development activities, promoting early and frequent feedback. Agile testing follows iterative cycles called sprints, each lasting from one to four weeks, where planning, development, testing, and review occur simultaneously. This methodology emphasizes collaboration among cross-functional teams and includes stakeholders in the feedback loop, allowing for quick adaptations to changing requirements. Agile testing is highly flexible, improving defect detection and ensuring a high-quality product. However, it requires strong communication, collaboration, and discipline to manage the iterative cycles effectively.

Spiral Model:



The Spiral testing methodology is a risk-driven process model that combines iterative development with the systematic aspects of the Waterfall model. It focuses heavily on risk analysis and mitigation throughout the development lifecycle. Each iteration, or spiral, consists of four key phases: planning, risk analysis, engineering and development, and evaluation. This model allows for iterative refinement, with each cycle building on the previous one, providing flexibility to incorporate changes and feedback. The risk management aspect helps in identifying and addressing potential issues early, reducing the chances of project failure. However, the Spiral model can be complex to manage and requires expertise in risk assessment and analysis, making it more suitable for large and critical projects.

Requirement Traceability Matrix (RTM):

Requirement Traceability Matrix (RTM) is a document that maps and traces user requirement with test cases. It captures all requirements proposed by the client and requirement traceability in a single document, delivered at the conclusion of the Software development life cycle. The main purpose of Requirement Traceability Matrix is to validate that all requirements are checked via test cases such that no functionality is unchecked during Software testing.

Types of Traceability Test Matrix:

In Software Engineering, traceability matrix can be divided into three major component as mentioned below:

Forward traceability: This matrix is used to check whether the project progresses in the desired direction and for the right product. It makes sure that each requirement is applied to the product and that each requirement is tested thoroughly. It maps requirements to test cases.

Backward or reverse traceability: It is used to ensure whether the current product remains on the right track. The purpose behind this type of traceability is to verify that we are not expanding the scope of the project by adding code, design elements, test or other work that is not specified in the requirements. It maps test cases to requirements.

Bi-directional traceability (Forward+Backward): This traceability matrix ensures that all requirements are covered by test cases. It analyzes the impact of a change in requirements affected by the defect in a work product and vice versa.

Key Components:

Requirement ID: A unique identifier for each requirement.

Requirement Description: A detailed description of the requirement.

Test Case ID: The identifier for the test case that verifies the requirement.

Test Case Description: A brief description of the test case.

Status: The status of the test case (e.g., Not Started, In Progress, Passed, Failed).

Process of Creating RTM:

Requirement Gathering: Collect all the requirements from stakeholders and documentation.

Test Case Development: Develop test cases based on the requirements.

Mapping Requirements to Test Cases: Link each requirement to one or more corresponding test cases.

Updating RTM: As the project progresses, update the RTM with the status of each test case.

Test Case in Software Testing:

A test case refers to the actions required to verify a specific feature or functionality in software testing. The test case details the steps, data, prerequisites, and postconditions necessary to verify a feature.

Standard Test Case Format:

- Test Case ID
- Test Scenario
- Test Steps
- Test Data
- Expected/Intended Results
- Actual Results
- Test Status – Pass/Fail

Test Case Example:

Test Case ID: TC001

Test Scenario: To authenticate a successful user login on gmail.com

Test Steps:

1. The user navigates to Gmail.com.
2. The user enters a registered email address in the 'email' field.
3. The user clicks the 'Next' button.
4. The user enters the registered password.
5. The user clicks 'Sign In.'

Test Data: username and password.

Expected/Intended Results: Once username and password are entered, the web page redirects to the user's inbox, displaying and highlighting new emails at the top.

Actual Results: As Expected

Test Status – Pass/Fail: Pass

Agile Framework:

Agile frameworks are methodologies designed to help teams and organizations manage work more efficiently and adapt to changes quickly. These frameworks promote iterative development, collaboration, and flexibility. Here's a detailed explanation of some of the most popular Agile frameworks:

1. Scrum

Overview: Scrum is one of the most widely used Agile frameworks. It emphasizes iterative progress through sprints, which are fixed-length periods (usually 2-4 weeks) where a set of tasks is completed.

Key Roles:

Product Owner: Represents the stakeholders and the voice of the customer. They are responsible for defining the features of the product and prioritizing the backlog.

Scrum Master: responsible for ensuring that the Scrum team adheres to Scrum principles, practices, and rules. They act as a servant leader and coach, facilitating collaboration and removing impediments to help the team achieve its goals.

Ceremonies:

Sprint Planning: The team plans the work for the upcoming sprint.

Daily Stand-Up: A short, daily meeting where team members discuss progress and obstacles.

Sprint Review: The team demonstrates what has been accomplished during the sprint.

Sprint Retrospective: The team reflects on the sprint and identifies areas for improvement.

Artifacts:

Product Backlog: The Product Backlog is an ordered list of everything that is known to be needed in the product. It is a dynamic and evolving document that provides a single source of requirements for any changes to be made to the product.

Sprint Backlog: The Sprint Backlog is a subset of the Product Backlog that the team commits to completing during a specific sprint. It includes a list of tasks that the team plans to work on and complete within the sprint.

The Product Backlog provides a long-term view of the product's requirements and priorities, while the Sprint Backlog focuses on the immediate tasks and goals for the current sprint. Together, they help the team maintain clarity, focus, and alignment with the overall product vision and goals.

2. Kanban

Overview: Kanban focuses on visualizing the flow of work, limiting work in progress (WIP), and maximizing efficiency. It is less prescriptive than Scrum and can be implemented without drastic changes to the existing process.

Key Elements:

Kanban Board: A visual tool that represents the workflow. It usually consists of columns such as "To Do," "In Progress," and "Done."

Work-In-Progress Limits: Limits the number of tasks in each stage to prevent bottlenecks.

Continuous Delivery: Work items are completed and delivered continuously.

Principles:

Visualize Work: Use the Kanban board to see the status of all tasks.

Limit WIP: Control the number of tasks being worked on simultaneously.

Manage Flow: Ensure smooth progression of tasks from start to finish.

Make Process Policies Explicit: Clearly define and communicate how work is done.

Implement Feedback Loops: Regularly review the process and make improvements.

Improve Collaboratively: Use data and team insights to improve the process continuously.

3. Extreme Programming (XP)

Overview: XP is a software development methodology that aims to improve software quality and responsiveness to changing customer requirements. It emphasizes technical excellence and collaborative work.

Key Practices:

Pair Programming: Two programmers work together at one workstation, continuously reviewing each other's code.

Test-Driven Development (TDD): Writing tests before writing the actual code to ensure functionality and facilitate refactoring.

Continuous Integration: Code changes are integrated into the main branch frequently to detect issues early.

Refactoring: Continuously improving the code without changing its external behavior.

Collective Code Ownership: Everyone can contribute to the codebase, and no single person owns any part of the code.

Sustainable Pace: Work at a pace that can be maintained indefinitely without burnout.

4. Lean

Overview: Lean focuses on delivering value to the customer by optimizing efficiency and eliminating waste. It originated from Lean manufacturing principles but has been adapted for software development.

Principles:

Eliminate Waste: Identify and remove anything that doesn't add value to the customer.

Build Quality In: Ensure quality at every stage of development.

Create Knowledge: Use feedback and learning to improve.

Defer Commitment: Make decisions as late as possible when more information is available.

Deliver Fast: Provide value quickly to get feedback sooner.

5. Crystal

Overview: Crystal is a family of methodologies designed to be adaptable to the size and criticality of the project. It focuses on people and their interactions over processes and tools.

Key Characteristics:

Human-Powered: Emphasizes the talents and collaboration of people.

Adaptable: Different versions (e.g., Crystal Clear, Crystal Orange) are suitable for different team sizes and project criticalities.

Frequent Delivery: Regularly deliver usable increments to gather feedback.

Reflective Improvement: Regularly reflect on the process and make necessary adjustments.

Principles:

Frequent Delivery: Deliver working software frequently.

Reflective Improvement: Continuously reflect and improve the process.

Osmotic Communication: Ensure information flows naturally within the team.

Personal Safety: Create an environment where team members can speak openly without fear.

Focus: Maintain focus on the most important tasks.

Agile frameworks offer various approaches to managing and executing projects, each with its own set of practices, roles, and principles. The choice of framework depends on the specific needs, culture, and goals of the organization. Regardless of the framework, the core Agile principles of iterative development, collaboration, and adaptability remain central.

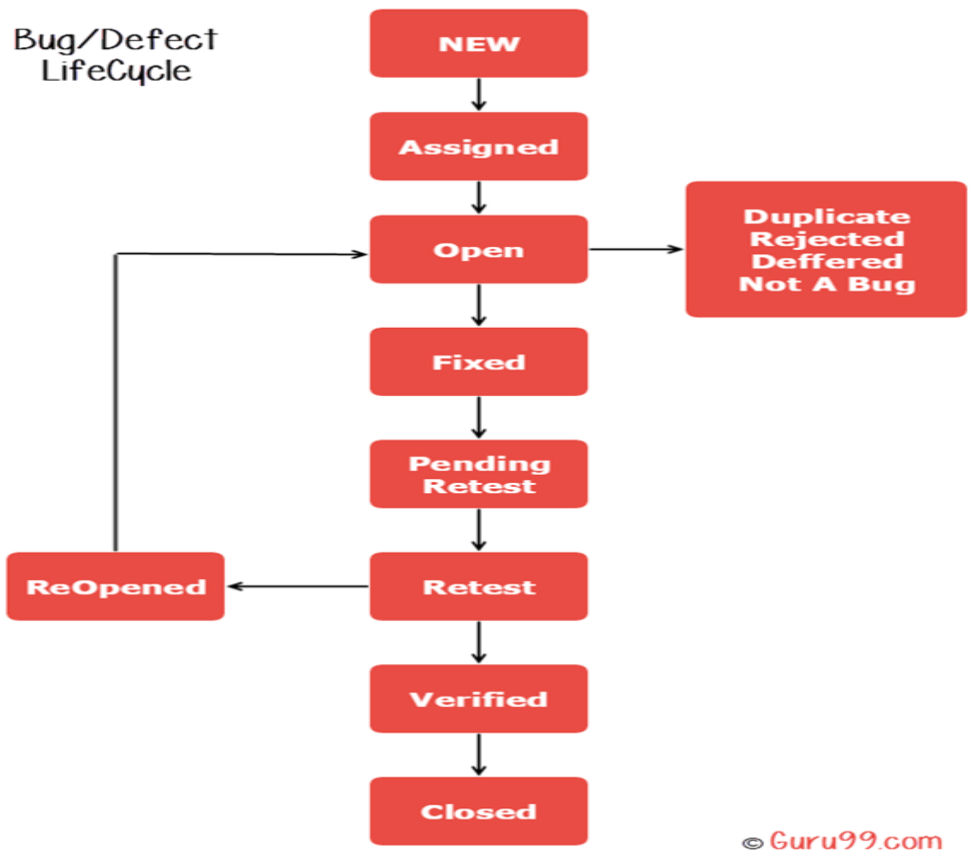
Defect Lifecycle Management:

Defect Life Cycle or Bug Life Cycle in software testing is the specific set of states that defect or bug goes through in its entire life. The purpose of Defect life cycle is to easily coordinate and communicate current status of defect which changes to various assignees and make the defect fixing process systematic and efficient.

Defect Status

Defect Status or Bug Status in defect life cycle is the present state from which the defect or a bug is currently undergoing. The goal of defect status is to precisely convey the current state or progress of a defect or bug in order to better track and understand the actual progress of the defect life cycle.

Defect States Workflow



New: When a new defect is logged and posted for the first time. It is assigned a status as NEW.

Assigned: Once the bug is posted by the tester, the lead of the tester approves the bug and assigns the bug to the developer team

Open: The developer starts analyzing and works on the defect fix

Fixed: When a developer makes a necessary code change and verifies the change, he or she can make bug status as "Fixed."

Pending retest: Once the defect is fixed the developer gives a particular code for retesting the code to the tester. Since the software testing remains pending from the testers end, the status assigned is "pending retest."

Retest: Tester does the retesting of the code at this stage to check whether the defect is fixed by the developer or not and changes the status to "Re-test."

Verified: The tester re-tests the bug after it got fixed by the developer. If there is no bug detected in the software, then the bug is fixed and the status assigned is "verified."

Reopen: If the bug persists even after the developer has fixed the bug, the tester changes the status to "reopened". Once again the bug goes through the life cycle.

Closed: If the bug is no longer exists then tester assigns the status "Closed."

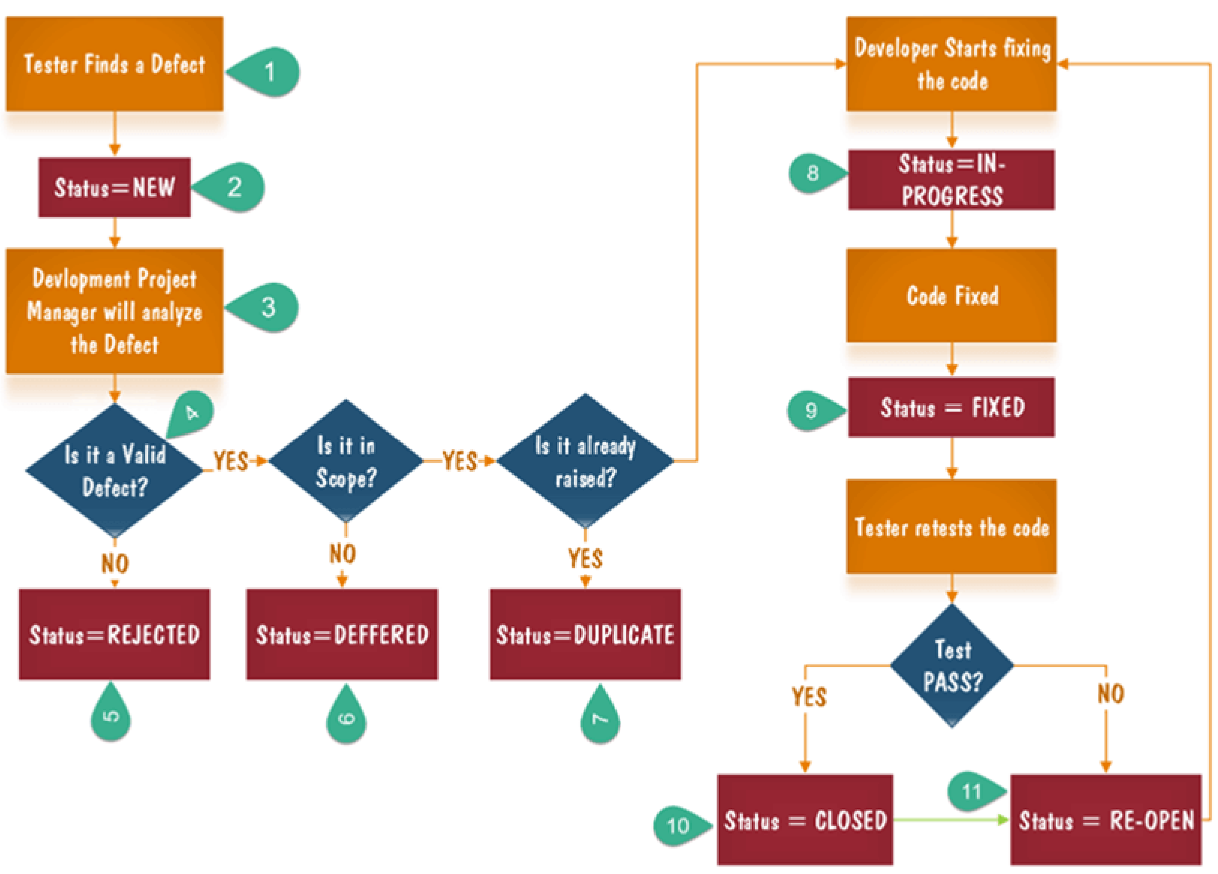
Duplicate: If the defect is repeated twice or the defect corresponds to the same concept of the bug, the status is changed to "duplicate."

Rejected: If the developer feels the defect is not a genuine defect then it changes the defect to "rejected."

Deferred: If the present bug is not of a prime priority and if it is expected to get fixed in the next release, then status "Deferred" is assigned to such bugs

Not a bug: If it does not affect the functionality of the application then the status assigned to a bug is "Not a bug".

Defect/Bug Life Cycle:



Example on how to write defect summary:

Defect Summary:

STAGE: Add to cart button disabled

Priority: Blocker

Assignee:

Defect Description:

Steps to reproduce:

Login to Amazon.in

Search for any product and navigate to its product detail page.

Observe the "Add to Cart" button.

Actual Result:

"Add to Cart" button is disabled and cannot be clicked.

Expected Result:

"Add to Cart" button should be enabled, allowing users to add products to their cart.