MANUAL TESTING

What is software?

Software is a collection of instructions, data, or computer programs that are used to run machines and carry out particular activities. It is the antithesis of hardware, which refers to a computer's external components. A device's running programs, scripts, and applications are collectively referred to as "software" in this context.
Eg: Gpay: used for payment
Fb ana insta: communication software
Amazon: ecommerce

What is software testing?

Software testing is the process of checking whether the quality, functionality, and performance of software meet expectations. It involves testers either interacting with the software manually or executing test scripts to identify bugs and ensure the software is defect-free.
Software testing is an art of gathering information through manual or automation means by making observation and comparing with actual requirement.
It is a process of checking whether actual requirement is same as the expected requirement.

7 Reasons why software testing is important?

1. Helps in saving money
2. Security
3. Quality of the product
4. Satisfaction of the customer
5. Enhancing the development process
6. Easy while adding new features
7. Determining the performance of the software

Types of Companies

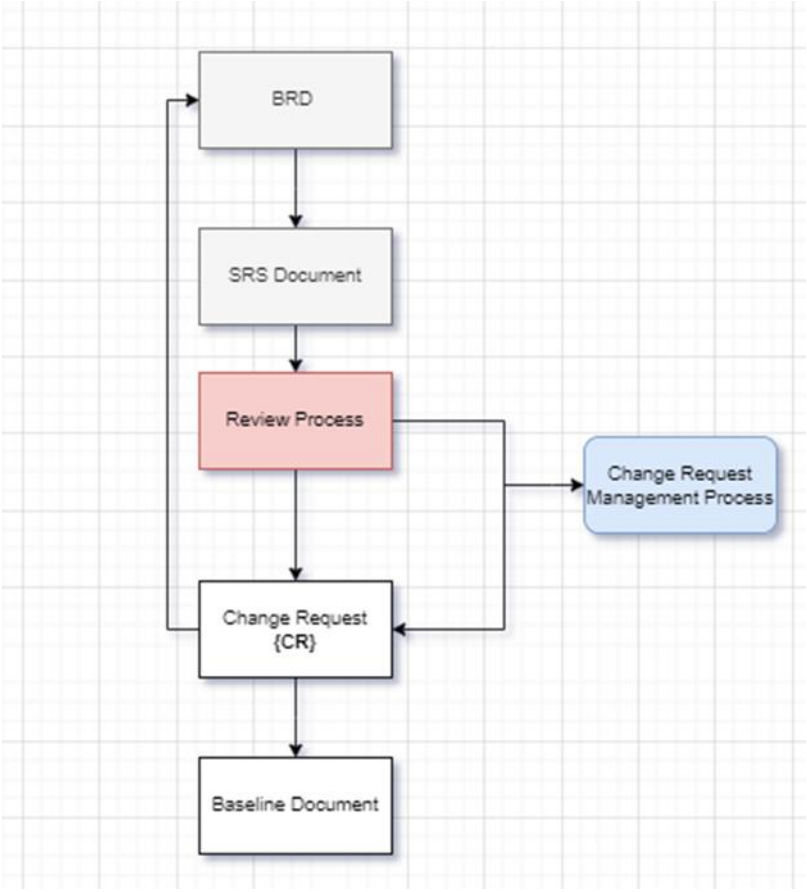
1. Product Based Company: eg: Swiggy, Zomato
2. Service Based Company: eg Infosys, TCS: They work on different products.

Different Types of Applications

1. Desktop applications: Applications installed in desktop. It will be installed by downloading an exe file.
2. Web based applications: Accessed using a browser. Eg: IRCTC, Gmail, GMeet.
3. Mobile applications: Downloaded from a play store or some other to a mobile.

Beginner Guides to Software Testing

1. Requirement analysis
2. Use case
3. Business Requirement Document (BRD)
4. System Requirement Specification (SRS)
5. Review and Baseline Process
6. Change Request (CR), Change Request Management Process
7. Baseline Requirement, Baseline Document
8. User Story (Summary, Description, Definition of Done (DOD), Acceptance Criteria)



REQUIREMENT ANALYSIS

Requirement analysis or requirements engineering is a process used to determine the needs and expectations of a new product. It involves frequent communication with the stakeholders and end users of the product to define expectations, resolve conflicts, and document all the key requirements.

Client's need or expectations in the product.

USE CASE

A use case is a text-based document that describes how one person interacts with a system to accomplish a particular goal. It's typically a list of steps written about this process from an individual's perspective. You can write a use case for various purposes, such as testing a software feature or creating a guide manual for customers.

Use cases are the first form of collecting requirements.

Eg: ecommerce websites.

Use Case 1: User Registration
Actor: New User
Main Flow:

1. User navigates to sign up page
2. User clicks on sign up button
3. User enters their personal information (user name, p/w, email)
4. System validates the information, ensuring the email is unique & p/w meets security criteria.
5. User submits the registration form
6. System sends a verification mail
7. User verifies the email & gain access to the ecommerce website.

Use Case 2: User Login
Actor: Existing User
Main Flow:

1. User navigates to sign up page
2. User clicks on login button
3. User enters valid username & password
4. User clicks on login button
5. System validates the username and password credentials are correct
6. User gain access to home page

Use Case 3: Product browsing & search
Actor: User
Main Flow:

1. User login to ecommerce website
2. User access the homepage or categories section
3. User search for a product using the search bar or filters by category, price, brand, etc.
4. System retrieves and displays the relevant products
5. User clicks on a product to view details

Use Case 4: Add to Cart & Manage Cart

Actor: User
Main Flow:

1. User navigates to homepage
2. User search for a product using the search bar
3. User clicks on add to cart on a product page
4. System adds the product to the cart and updates the cart count
5. User can view the cart, update quantities or remove item
6. System updates the cart details accordingly.

Assignment:
Checkout & order placement - Aneesh
Order tracking and delivery - Keerthana
Order cancellation & refund - Shabina
User review & ratings - Sumana
Amazon Pay – Akhila

BUSINESS REQUIREMENT DOCUMENT (BRD)
Business analyst collects all the requirements needed for the client and make a document this document is called as Business Requirement Document.
High level document.
From client to Business Analyst.

SYSTEM REQUIREMENTS SPECFICATIONS (SRS)
This document will have the total requirement of the technical aspects needed for software development

- Business Analyst to Software architect
- This document is made by the software architect
- Technical conversion of BRD document to the terms of developers

REVIEW PROCESS
Review process is a process where the BRD and SRS documents are reviewed by the client and verified whether all the requirements are met. If not, it will be moved to Change Request Process. Else, it will be moved to Baseline Process.

CHANGE REQUEST (CR)
A Change Request (CR) is a formal proposal to modify a system requirement, design, or process after the initial specifications have been approved.

CHANGE REQUEST MANAGEMENT PROCESS
The Change Request Management Process ensures that every CR is properly evaluated, documented, and implemented without negatively impacting the system.

It includes steps like submission, impact analysis, approval, implementation, testing, and final deployment.

BASELINE REQUIREMENTS

Baseline Requirement refers to a requirement that has been formally reviewed and approved, serving as a reference point for development and testing. Changes to baseline requirements follow a controlled process.

BASELINE PROCESS

The Baseline Process is the act of finalizing a document or software component after reviews. Once a document is baselined, any changes require formal approval through a change management process.

BASELINE DOCUMENT

Baseline Document is a finalized version of a project document (like SRS or test plan) that has been reviewed, approved, and placed under configuration control. It serves as an official reference for the project team.

USER STORY

A user story is a requirement for any functionality or feature which is written down in one or two lines and max up to 5 lines. A user story is usually the simplest possible requirement and is about one and only one functionality (or one feature).
Eg: facebook - registration

Summary	Definition of Done (DOD)	Acceptance Criteria
Functionality of user registration.	As a user I want to create a new account & access facebook.	1. There should be a field for entering First Name. 2. There should be a field for entering Surname. 3. There should be labels mentioning the field: • First Name • Surname • DOB • Gender • Mobile number/Email address • Password

		4. There should be drop downs available for selecting DOB 5. There will be radio buttons for selecting gender. 6. There should be field for entering mobile no/email address 7. Password field. 8. There should be a submit button. 9. All fields should be mandatory and don't accept any empty submissions. 10. Display validation message for empty submissions. 11. First name and Surname field should accept only characters and should not accept any numbers or special characters. 12. The length of first name and surname field should not exceed 8 characters. 13. The user should have minimum age of 18yrs to create a Facebook account, else display user is not eligible for creating account. 14. For gender, user should be able to select only one checkbox at a time. 15. Mobile no/email field should accept either

		mobile number or email. 16. After clicking sign up button, validate all field are having valid data. 17. The password field should accept a combination of character, number and special character. 18. The password length should be between 8 to 12 characters. 19. There should be a button for view and hide password. 20. There should be sign up button for submitting all the details. 21. If the email or phone number is already existing, show an alert message to enter new phone no/email address.

SDLC (Software Development Life Cycle)

SDLC is a process followed for software building within a software organization. SDLC consists of a precise plan that describes how to develop, maintain, replace, and enhance specific software. The life cycle defines a method for improving the quality of software and the all-around development process.

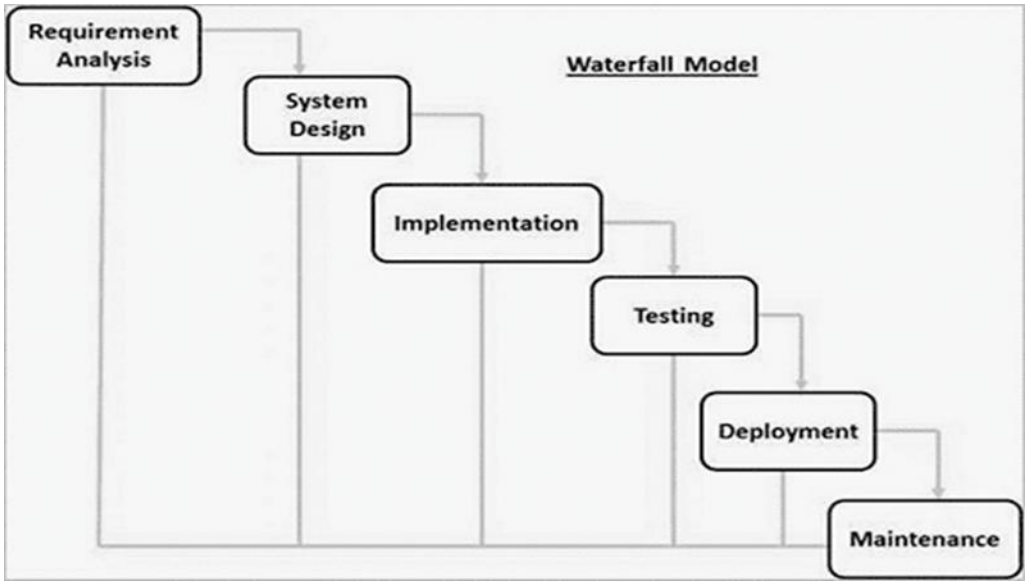
SDLC is an abbreviation of Software Development Life Cycle. SDLC is series of steps that offers a defined model for the development and lifecycle management of an application.

1. Waterfall model
2. V-Model
3. Iterative model
4. Incremental Model

5. Agile Model

Different phases of SDLC: Requirement Analysis, System Design, Implementation, Testing, Deployment and Maintenance.

WATERFALL MODEL



Waterfall approach was the first SDLC Model to be used widely in Software Engineering to ensure success of the project. In "The Waterfall" approach, the whole process of software development is divided into separate phases. In this Waterfall model, typically, the outcome of one phase acts as the input for the next phase sequentially.

- **Requirement Gathering and analysis** – All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification document.
- **System Design** – The requirement specifications from first phase are studied in this phase and the system design is prepared. This system design helps in specifying hardware and system requirements and helps in defining the overall system architecture.
- **Implementation** – With inputs from the system design, the system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality, which is referred to as Unit Testing.
- **Integration and Testing** – All the units developed in the implementation phase are integrated into a system after testing of each

unit. Post integration the entire system is tested for any faults and failures.

- **Deployment of system** – Once the functional and non-functional testing is done; the product is deployed in the customer environment or released into the market.
- **Maintenance** – There are some issues which come up in the client environment. To fix those issues, patches are released. Also to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment.

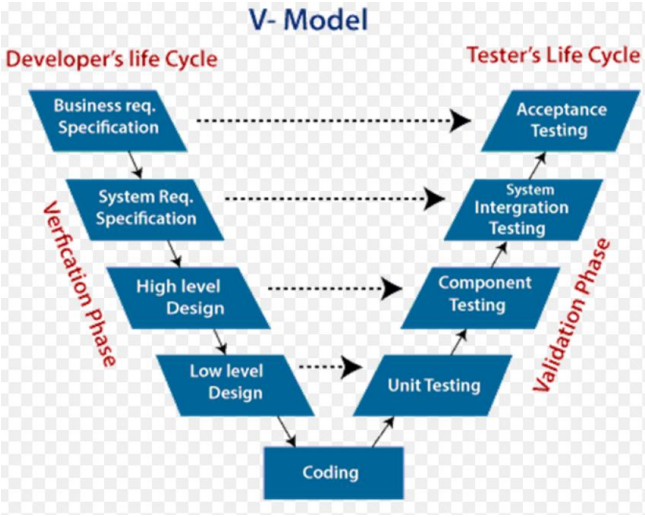
Waterfall Model – Advantage

- Simple and easy to understand and use
- Easy to manage due to the rigidity of the model. Each phase has specific deliverables and a review process.
- Phases are processed and completed one at a time.
- Works well for smaller projects where requirements are very well understood.
- Clearly defined stages.
- Well understood milestones
- Easy to arrange tasks
- Process and results are well documented.

Waterfall Model – Disadvantage

- No working software is produced until late during the life cycle
- High amounts of risk and uncertainty
- Not a good model for complex and object-oriented projects.
- Poor model for long and ongoing projects.
- Not suitable for the projects where requirements are at a moderate to high risk of changing. So, risk and uncertainty are high with this process model
- It is difficult to measure progress within stages
- Cannot accommodate changing requirements.
- Adjusting scope during the life cycle can end a project.

V-MODEL



V-Model also referred to as the **Verification and Validation Model**. In this, each phase of SDLC must complete before the next phase starts. It follows a sequential design process same as the waterfall model. Testing of the device is planned in parallel with a corresponding stage of development.

Verification:
It involves a static analysis method (review) done without executing code. It is the process of evaluation of the product development process to find whether specified requirements meet.

Validation:
It involves dynamic analysis method (functional, non-functional), testing is done by executing code. Validation is the process to classify the software after the completion of the development process to determine whether the software meets the customer expectations and requirements. So, V-Model contains Verification phases on one side and Validation phases on the other side. Verification and Validation process is joined by coding phase in V-shape. Thus, it is known as V-Model.

There are the various phases of Verification
Phase of V-model:

1. Business requirement analysis: This is the first step where product requirements understood from the customer's side. This phase contains detailed communication to understand customer's expectations and exact requirements.

2. System Design: In this stage system engineers analyze and interpret the business of the proposed system by studying the user requirements document.

3. High level Design: The baseline in selecting the architecture is that it should understand all which typically consists of the list of modules, brief functionality of each module, their interface relationships, dependencies, database tables, architecture diagrams, technology detail, etc. The integration testing model is carried out in a particular phase.

4. Low level Design: In the module design phase, the system breaks down into small modules. The detailed design of the modules is specified, which is known as Low-Level Design.

5. Coding Phase: After designing, the coding phase is started. Based on the requirements, a suitable programming language is decided. There are some guidelines and standards for coding. Before checking in the repository, the final build is optimized for better performance, and the code goes through many code reviews to check the performance.

6. Unit testing is a type of software testing where individual units or components of a software are tested. The purpose is to validate that each unit of the software code performs as expected. Unit Testing is done during the development (coding phase) of an application by the developers.

7. Component testing is a software testing technique that involves testing each individual component of a software application separately without integrating it with other component

8. System Integration Testing is defined as a type of testing where software modules are integrated logically and tested as a group. A typical software project consists of multiple software modules, coded by different programmers. The purpose of this level of testing is to expose defects in the interaction between these software modules when they are integrated

9. Acceptance testing involves testing the system’s Functional and Non-functional aspects, such as performance, security, usability, accessibility, compatibility, and reliability. (It is a high-level testing, that can be done by client itself or some high level employee of our company). Depending on the system’s complexity. it contains alpha testing and beta testing:

● Alpha Testing: This phase is conducted by internal testers, often within the organization, to validate the product's quality and functionality. It uses both white box and black box testing techniques to identify bugs and ensure the software is stable and working as intended. Alpha testing is performed in a controlled environment and can last for several weeks, with multiple test cycles.

● Beta Testing: This phase involves real users outside the organization who test the software in their own environments. Beta testing focuses on usability, functionality, security, and reliability, and it helps gather feedback that can be used to improve the product before its official release. Beta testing is typically shorter, lasting a few weeks, and it is unstructured, allowing users to use the software freely and provide feedback.

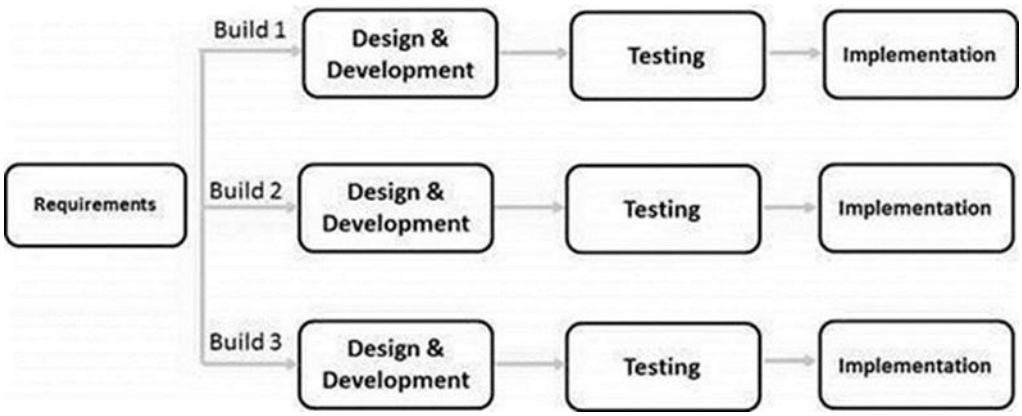
Advantages of V-Model

1. Easy to Understand.
2. Testing Methods like planning, test designing happens well before coding.
3. This saves a lot of time. Hence a higher chance of success over the waterfall model.
4. Avoids the downward flow of the defects.
5. Works well for small plans where requirements are easily understood.

Disadvantages of V-Model

1. Very rigid and least flexible.
2. Not a good for a complex project.
3. Software is developed during the implementation stage, so no early prototypes of the software are produced.
4. If any changes happen in the midway, then the test documents along with the required documents, has to be updated.

ITERATIVE MODEL



Iterative process starts with a simple implementation of a subset of the software requirements and iteratively enhances the evolving versions until the full system is implemented. At each iteration, design modifications are made and new functional capabilities are added. The basic idea behind this method is to develop a system through repeated cycles (iterative) and in smaller portions at a time (incremental).

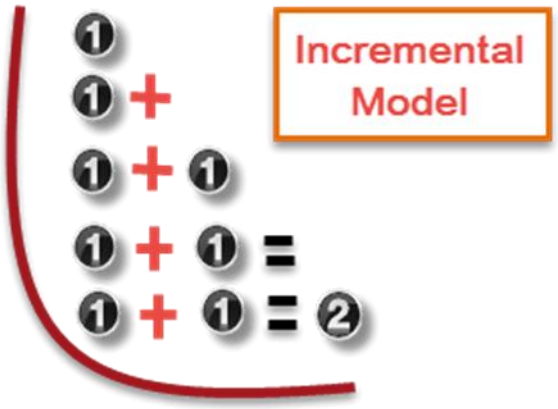
Advantage of Iterative Model

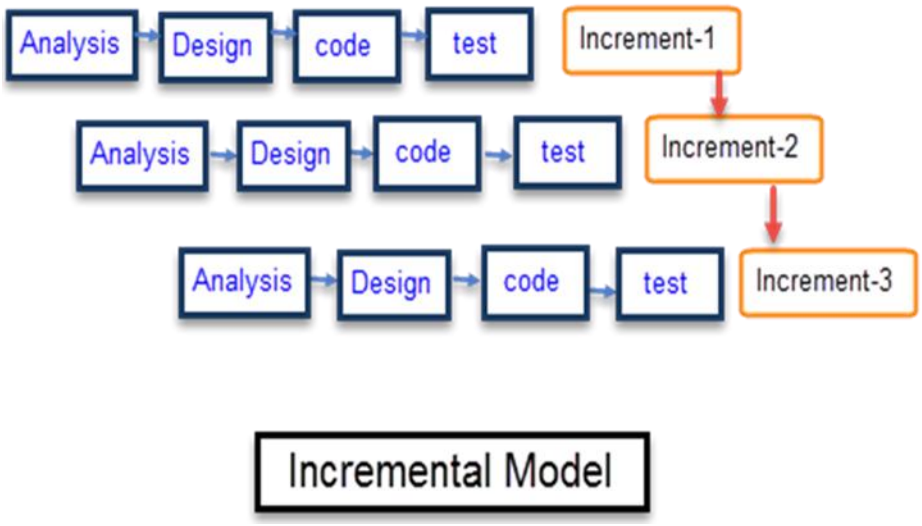
- Some working functionality can be developed quickly and early in the life cycle.
- Results are obtained early and periodically.
- Parallel development can be planned.
- Testing and debugging during smaller iteration is easy.
- Initial Operating time is less.

Disadvantages of Iterative Model

- More resources may be required.
- Although cost of change is lesser, but it is not very suitable for changing requirements. • More management attention is required.
- Not suitable for smaller projects.

INCREMENTAL MODEL





Incremental Model

Incremental Model is a process of software development where requirements are broken down into multiple standalone modules of software development cycle. Incremental development is done in steps from analysis design, implementation, testing/verification, maintenance.

Each iteration passes through the requirements, design, coding and testing phases. And each subsequent release of the system adds function to the previous release until all designed functionality has been implemented.

Advantages of Incremental Model

• The software will be generated quickly during the software life cycle

• It is flexible and less expensive to change requirements and scope

• Throughout the development stages changes can be done

• This model is less costly compared to others

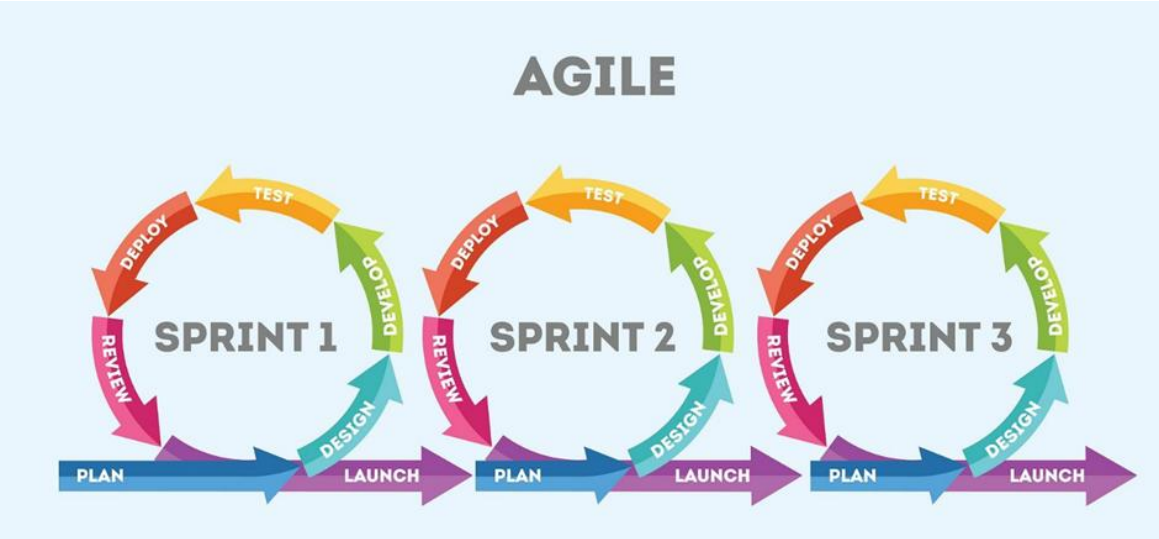
Disadvantages Incremental Model

• It requires a good planning designing

• Problems might cause due to system architecture as such not all requirements collected up front for the entire software lifecycle

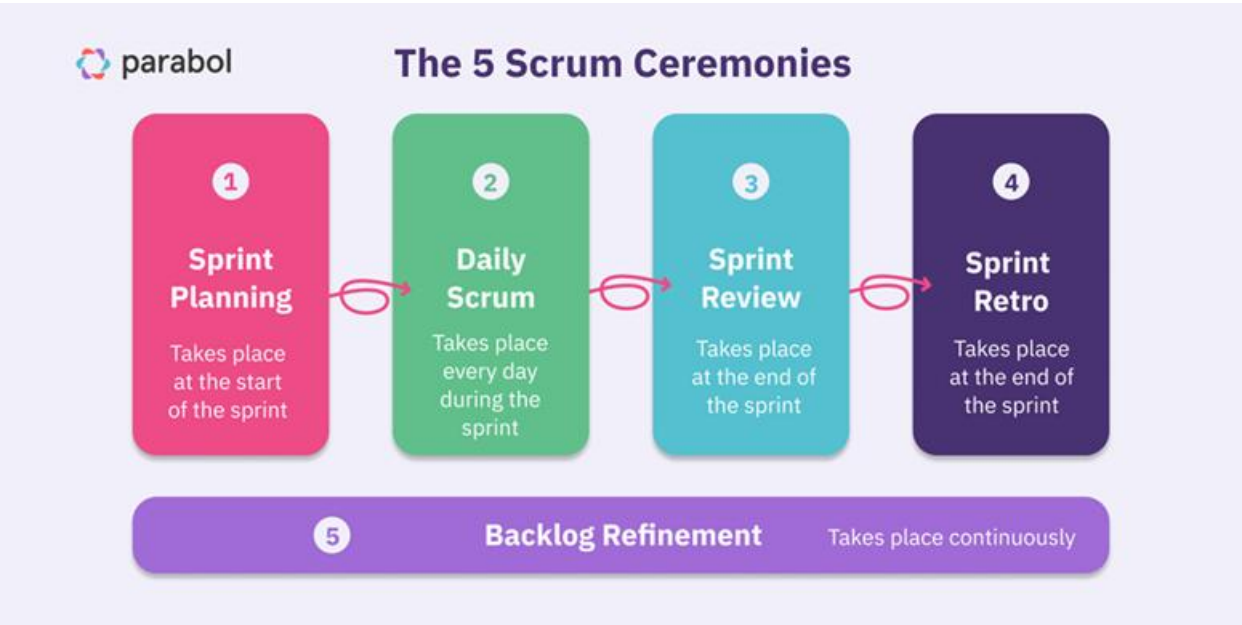
• Each iteration phase is rigid and does not overlap each other

AGILE MODEL



The agile methodology is an iterative and incremental approach to project management that emphasizes flexibility, collaboration, and continuous improvement. It is a popular framework for software development, but it can be applied to other areas as well. The agile methodology is based on the principles of iterative development, where the project is broken down into smaller, manageable chunks, and each chunk is developed and delivered in a series of iterations.

Agile Scrum Ceremonies



1. SPRINT PLANNING:
   Plan and Estimate -This phase consists of processes related to planning and estimating tasks, which include Create User Stories, Approve, Estimate, and Commit User Stories, Create Tasks, Estimate Tasks, and Create Sprint Backlog.

2. DAILY SCRUM:
   Daily scrums are quick meetings held each day at the same time for members of the product development team working on a particular sprint. The team collectively reviews the progress made toward achieving the Sprint Goal.

3. SPRINT REVIEW:
   A sprint review is a collaborative meeting that is typically held at the end of every sprint. This is when the team runs through work items they completed during the sprint or iteration. A sprint review ensures key stakeholders are up to date, and it enables them to provide feedback.

4. SPRINT RETROSPECTIVE:
   As a tester, he will figure out what went wrong and what went right in the current sprint. As a tester, he identifies lessons learned and best practices.

5. BACKLOG REFINEMENT:
   Backlog refinement (formerly known as backlog grooming) is when the product owner and some, or all, of the rest of the team review items on the backlog to ensure the backlog contains the appropriate items, that they are prioritized, and that the items at the top of the backlog are ready for delivery.

Check who all are included in a Scrum ie, scrum members: Product Owner, Scrum Master, Scrum Team (Development Team, Testing team, etc).

TEST CASE AND TEST SCENARIO

Test Scenario:

A Test Scenario is defined as any functionality that can be tested. It is also called Test Condition or Test Possibility. As a tester, you should put yourself in the end user's shoes and figure out the real-world scenarios and use cases of the Application Under Test. **Test scenarios are focused on what to test.**

Test Case:

Test case, **they are focused on how to test**. Test cases contain definite test steps, data, and expected outcomes for testing all the features of the software under test.

Assignment (User Story)

1. Hospital Management System – Write a user story for patient appointment scheduling. - Akhila

2. Learning Management System (LMS) – Write a user story for enrolling in a course. - Sumana

3. Food Delivery Application – Write a user story for placing an order. - Anish 4. Ride-Sharing App – Write a user story for booking a ride. – Keerthana

TEST TASK AND ACTIVITIES

1. Coding
2. Code Review
3. Unit Testing
4. Test Case Preparation
5. Test Case Review
6. Test Data Preparation (Demo Atm Card, Demo User Names, Passwords)
7. Test Case Execution

1. Coding – The process where developers write software code based on design specifications and requirements. It involves implementing functionalities, fixing bugs, and ensuring the system works as intended.

2. Code Review – A systematic examination of code by peers or leads to identify defects, improve code quality, ensure adherence to coding standards, and enhance maintainability before merging into the main codebase.

3. Unit Testing – A type of testing where individual components or functions of the software are tested in isolation to verify they work correctly. Developers usually write unit tests using frameworks like JUnit (Java), PyTest (Python), or NUnit (.NET).

4. Test Case Preparation – The process of designing detailed test cases based on requirements and specifications to verify different functionalities, edge cases, and user scenarios of the application.

5. Test Case Review – A quality check process where test cases are reviewed by peers, leads, or business analysts to ensure accuracy, completeness, and proper test coverage before execution.

6. Test Data Preparation – Creating necessary data for testing, such as demo ATM card details, usernames, and passwords, to simulate real-world scenarios while ensuring privacy and security compliance.

7. Test Case Execution – Running the prepared test cases on the software to validate its functionality, find defects, and ensure it meets the expected requirements. Execution can be manual (by testers) or automated (using tools like Selenium, JUnit, or TestNG).

7 PRINCIPLES OF TESTING

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

Testing shows the presence of defects:

Testing reveals the presence of defects but cannot guarantee the absence of defects. This principle emphasizes the importance of identifying and fixing defects early in the development process.

Exhaustive testing is impossible:

It is impractical to test every possible combination of inputs and conditions due to the vast number of variables. Instead, testers should prioritize scenarios based on risk and business impact.

Early Testing:

Initiating testing early in the software development lifecycle is crucial for cost-effectiveness and defect prevention. Early detection of defects allows for more efficient and less costly fixes.

Defect Clustering:

Defect clustering in software testing refers to the phenomenon where most of the defects are concentrated in a few select areas of an application, rather than being evenly distributed throughout the system. This principle is based on the Pareto principle, which suggests that approximately 20% of the modules account for 80% of the bugs. This clustering can be caused by various factors, such as legacy code that is prone to breaking, newer features that undergo frequent changes, or complex third-party integrations.

Pesticide Paradox:

Repeatedly using the same tests without variation can lead to a situation where the same tests no longer find new defects. To avoid this, testers should regularly update and diversify their test cases.

Testing is context dependent:

Different types of software require different testing approaches. The testing strategy should be tailored to the specific context and requirements of the software being developed.

Absence of error fallacy:

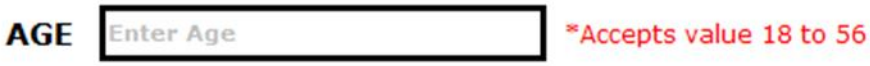
A software system can be free of detected errors but still fail to meet user needs or business requirements. Testing should not only focus on finding defects but also on ensuring the software meets user expectations and business goals.

TEST CASE DESIGN TECHNIQUES (Black Box Design Techniques)

1. Boundary Value Analysis
2. Equivalence Partition
3. Decision Table
4. State Transition
5. Error Guessing

BOUNDARY VALUE ANALYSIS

Invalid test cases	Valid test cases	Invalid test cases
11, 13, 14, 15, 16, 17	18, 19, 24, 27, 28, 30	31, 32, 36, 37, 38, 39



BOUNDARY VALUE ANALYSIS		
Invalid (min -1)	**Valid (min, +min, -max, max)**	**Invalid (max +1)**
17	18, 19, 55, 56	57

Focus on the boundaries of the input domain. Create test cases that test values at the lower and upper boundaries and just inside and outside those boundaries.

EQUIVALENCE PARTITION

EQUIVALENCE CLASS PARTITIONING (ECP)		
Invalid equivalence class	Valid equivalence class	Invalid equivalence class
< 10 digits	10 digits	> 10 digits

Divide the input domain of a function into classes or partitions of equivalent inputs. Create test cases that represent each partition, covering both valid and invalid data.

DECISION TABLE

Eg 1:

Email (condition1)	T	T	F	F
Password (condition2)	T	F	T	F
Expected Result (Action)	Account Page	Incorrect password	Incorrect email	Incorrect email

Eg: Create a decision table that captures the combinations of input conditions and their corresponding actions. Develop test cases to cover all possible combinations.

Interpretation:

• Case 1 – The username and password both were correct, and the user navigated to the homepage

• Case 2 – Username was correct, but the password was wrong. The user is shown an error message.

• Case 3 – Username was wrong, but the password was correct. The user is shown an error message.

• Case 4 – Username and password both were wrong. The user is shown an error message.

Eg 2:

2. **Shipping Rate Calculation**:

Condition 1: Package Weight	Condition 2: Destination
Less than 1kg	Domestic
Less than 1kg	International
Between 1kg and 5kg	Domestic
Between 1kg and 5kg	International
Greater than 5kg	Domestic
Greater than 5kg	International

Keep weight static & check rates for domestic & international.

Eg 3:

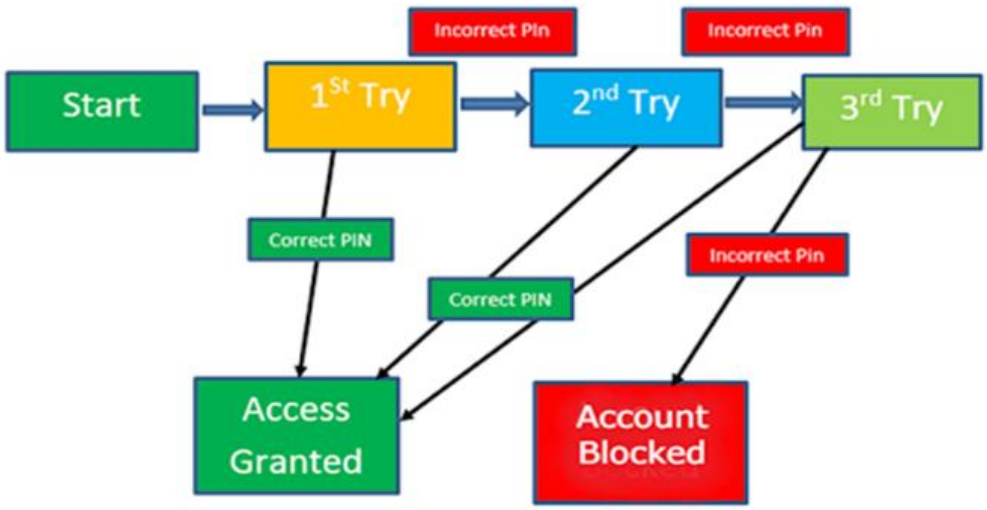
3. **Discount Calculation**:

Condition 1: Purchase Amount	Condition 2: Membership Status
Less than $50	Regular
Less than $50	Premium
$50 - $100	Regular
$50 - $100	Premium
Greater than $100	Regular
Greater than $100	Premium

Actions/Outcomes:

- Apply no discount
- Apply 5% discount for regular members
- Apply 10% discount for premium members

STATE TRANSITION



Suitable for systems with distinct states or modes. Create test cases to transition between states, including valid and invalid state transitions.

2. Traffic Light System:

States:
- Green Light
- Yellow Light
- Red Light

Transitions:
● From Green Light to Yellow Light after a certain time interval.
● From Yellow Light to Red Light after a certain time interval.
● From Red Light to Green Light after a certain time interval or on user command.

Test Cases:
● Test transition from green to yellow at the appropriate time.
● Test transition from yellow to red at the appropriate time.
● Test manual transition from red to green.

ERROR GUESSING

Error guessing is an informal testing technique where testers rely on their experience, intuition, and domain knowledge to identify potential defects in software applications that may not be captured by formal test cases or specifications. It involves guessing where errors or bugs might exist based on experience with similar systems or applications, common pitfalls, or understanding of user behaviour and expectations.

ALPHABETS OF MANUAL TESTING

1. Test Scenario
2. Test Case Preparation
3. Test Case Template & Test Case Data
4. Test Case Design Techniques
5. Error Bug Defect Failure
6. Test Task & Activities

Error Bug Defect Failure

Error

An error is a mistake made by a developer in the code.

Its main causes are:

• Negligence: Carelessness often leads to errors

• Miscommunication: An unclear feature specification or its improper interpretation by developers could lead to slips.

• Inexperience: Inexperienced developers often miss out on essential details, leading to faults down the line.

• Complexity: Intricate algorithms can cause developers to make mistakes in their coding logic.

Bug

In software testing, a bug is a deviation from the customer requirement, in simple language, we can say **deviation between the expected result and the actual result in an application or in a module that is found by the testing team during the testing period.**

Defect

If the functionality of **an application is not working as per the customer's requirement is known as a defect. It is found during the development phase while unit testing.** Giving wrong input may lead to a defect or Any code error may lead to a defect.

Failure

Once the software is completed and delivered to the customer **if the customer finds any issue in the software, then it is the condition of failure of the software.** In other words, if an end user finds an issue in the software, then that particular issue is called a failure.

Reasons:

• Missing test cases

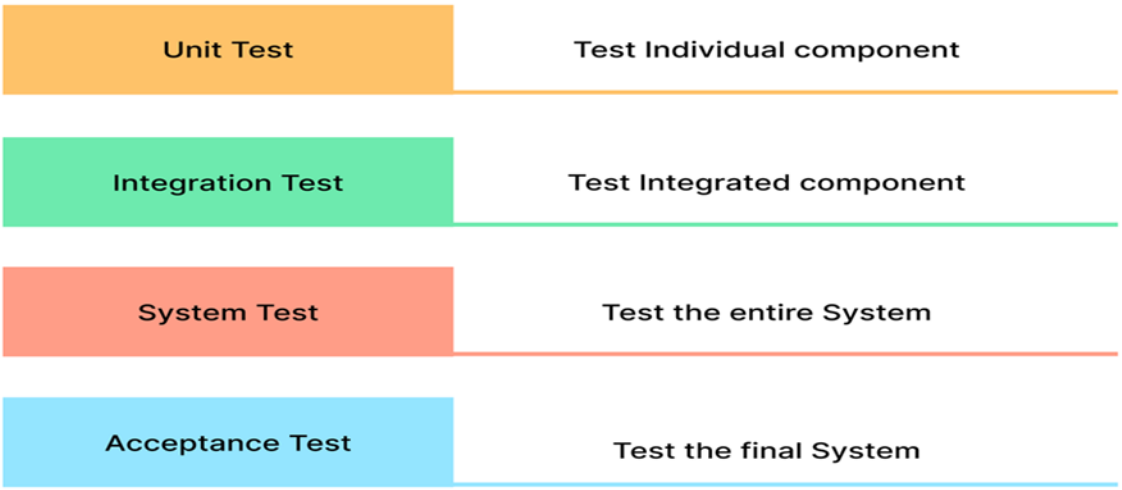
• Test cases not executed by the tester

• Improper execution of test cases

• Code changed after testing

LEVELS OF TESTING

1. Unit Testing
2. Integration Testing
3. System Testing
4. Acceptance Testing



Unit testing is done at the code level, where each component is tested individually to ensure their impartiality and analyse their functionality.

Integration testing enables software testers to test group units integrated into a system or subsystems; it helps identify any bugs or issues arising from coding errors or integrations between modules. It is possible to automate integration testing.

System testing is performed on an integrated environment comprising the whole application, where all components are assessed against specific business requirements.

Acceptance testing involves testing the system's Functional and Non-functional aspects, such as performance, security, usability, accessibility, compatibility, and reliability. Depending on the system's complexity. it contains alpha testing and beta testing

● Alpha Testing: This phase is conducted by internal testers, often within the organization, to validate the product's quality and functionality. It uses both white box and black box testing techniques to identify bugs and ensure the software is stable and working as intended. Alpha testing is performed in a controlled environment and can last for several weeks, with multiple test cycles.

● Beta Testing: This phase involves real users outside the organization who test the software in their own environments. Beta testing focuses on usability, functionality,

security, and reliability, and it helps gather feedback that can be used to improve the product before its official release. Beta testing is typically shorter, lasting a few weeks, and it is unstructured, allowing users to use the software freely and provide feedback.

ENVIRONMENTS

Example:

Dev Environment – Developers – Developing & Dev Testing (Unit Testing)
url: www.dev.amazon.com

QA Environment – Testers – Functional Testing
url: www.qa.amazon.com

Staging Environment - Testers – Non-Functional Testing
(Here we check the compatibility & performance, volume testing, security, etc)
url: www.p1.amazon.com

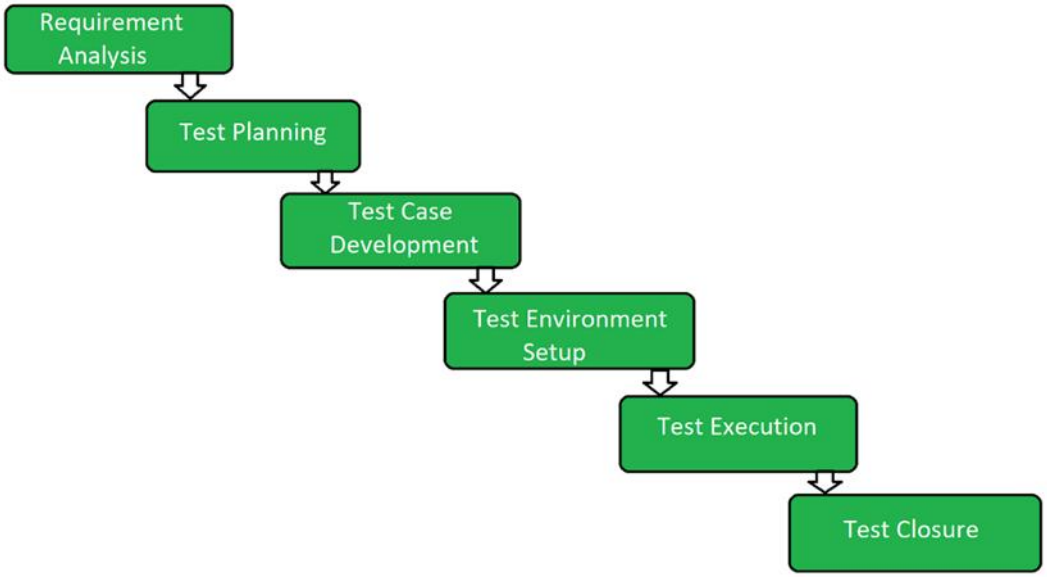
Pre-production/ UIT Environment – Client – Business Validation
(Here it will be alpha & beta testing. It is a final round testing)
url: www.p2.amazon.com

Production Environment – User
(Real time customers)
url: www.amazon.com

Let us look an example with versions.
Version-1.01 – Video Call (Let it be live i.e., Production Environment)
Version-1.02 – Disappear Message (Pre-production Environment)
Version-1.03 – Call Recording Option (Staging Environment)
Version-1.04 – Sticker Creation (QA Environment)
Version-1.05 – Status Upload (And it will be in Dev Environment)
Version-1.06 – Screen Sharing (It will be in Product Backlog Stage)

When Disappear Message moves to live stage, then,
Version-1.02 – Disappear Message (Production Environment)
Version-1.03 – Call Recording Option (Pre-Production Environment)
Version-1.04 – Sticker Creation (Staging Environment)
Version-1.05 – Status Upload (QA Environment)
Version-1.06 – Screen Sharing (Dev Environment)

STLC (Software Testing Life Cycle)



The Software Testing Life Cycle (STLC) is a systematic process that outlines specific stages involved in testing software to ensure its quality and functionality. Each phase has distinct objectives and deliverables, collectively contributing to the delivery of a reliable software product. The primary phases of the STLC include:

1. **Requirement Analysis**: In this initial phase, the testing team reviews and analyses the software requirements from a testing perspective to identify testable aspects. The goal is to ensure a clear understanding of the requirements to design effective test strategies.

2. **Test Planning**: Based on the requirements analysis, the test strategy is defined in this phase. Activities include determining the scope of testing, resource allocation, scheduling, and selecting appropriate testing tools. The outcome is a comprehensive test plan that guides the testing process.

3. **Test Case Development**: Here, detailed test cases are designed, including the preparation of test data. Each test case outlines specific conditions and expected results to validate the software's functionality against the requirements.

4. **Test Environment Setup**: This phase involves configuring the necessary hardware and software environments where testing will be conducted. It ensures that the test environment mirrors the production environment to identify potential issues accurately.

5. **Test Execution**: With the environment set up and test cases prepared, the testing team executes the tests. Any defects or discrepancies between expected and actual results are logged for further analysis.

6. **Test Closure**: After test execution, the team evaluates the testing process, documents findings, and assesses the overall quality of the software. This phase includes preparing test summary reports and identifying lessons learned to improve future testing cycles.

STORY POINT ESTIMATION

Story point	Amount of effort required	Amount of time required	Task complexity	Task risk or uncertainty
1	Minimum effort	A few minutes	Little complexity	None
2	Minimum effort	A few hours	Little complexity	None
3	Mild effort	A day	Low complexity	Low
5	Moderate effort	A few days	Medium complexity	Moderate
8	Severe effort	A week	Medium complexity	Moderate
13	Maximum effort	A month	High complexity	High

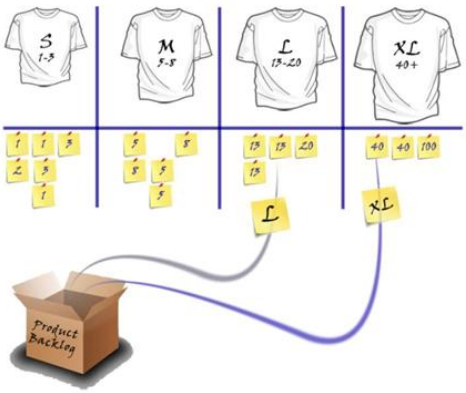
A user story point is a unit of measurement used in Agile project management to estimate the effort required to complete a user story. It is a way to quantify the complexity and difficulty of a task, allowing teams to prioritize and plan their work more effectively.

Story points are usually estimated during a planning meeting, where the team discusses and agrees on the effort required to complete a user story. The team members assign a number of story points to each user story based on their understanding of the task's complexity, uncertainty, and difficulty.

PLANNING POKER

Planning Poker: A popular technique where team members use numbered cards to vote on the story points of a task. The numbers typically follow the Fibonacci sequence (1, 2, 3, 5, 8, 13, 20, 40, 100). After discussion, each member privately selects a card representing their estimate. The team discusses any discrepancies and repeats the process until consensus is reached.

T-SHIRT SIZING



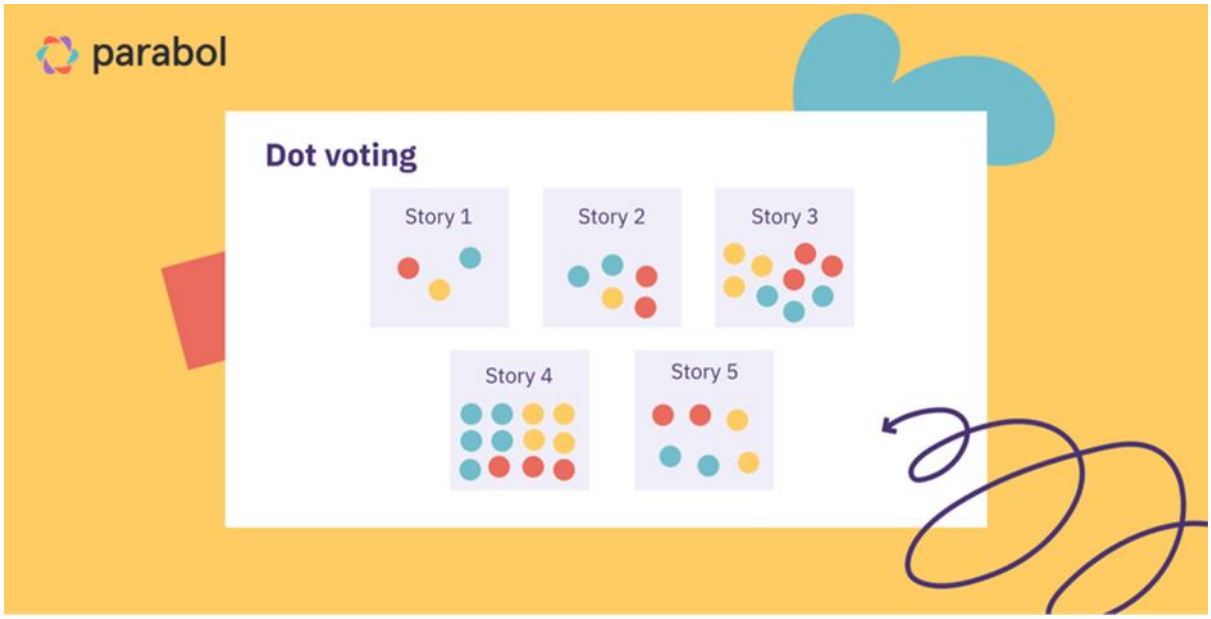
T-Shirt Sizing: This method uses sizes (S, M, L, XL) to estimate the effort required for tasks. It's simpler than planning poker and can be quicker for teams to use. Each size corresponds to a range of story points, such as S for 1-2 points, M for 3-5 points, L for 8-13 points, and XL for 20-40 points.

FIBONACCI SERIES

Teams use the Fibonacci sequence (1, 2, 3, 5, 8, 13, 20, 40, 100) to estimate the effort required for tasks. The sequence is used because the numbers are spaced far apart, making it easier for the team to reach a consensus on which number to use.

DOT VOTING

In this technique, team members vote on the story points of tasks by placing dots on a chart or board. Each dot represents a story point. This method is useful for prioritizing tasks and can be combined with other techniques like planning poker or T-shirt sizing.



FUNCTIONAL & NON-FUNCTIONAL TESTING

Functional testing is a type of testing that seeks to establish whether each application feature works as per the software requirements. Each function is compared to the corresponding requirement to ascertain whether its output is consistent with the end user's expectations.

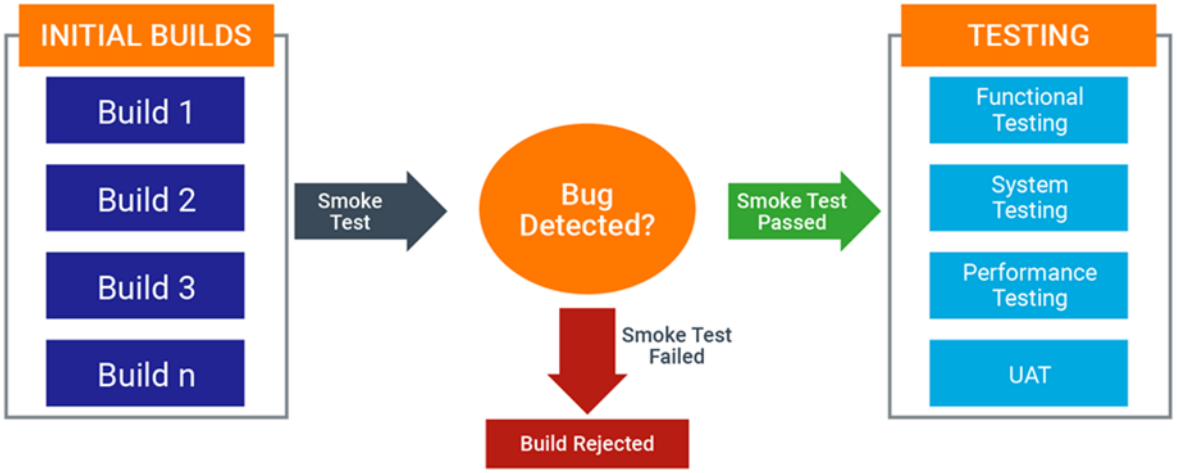
Non-functional testing is a type of software testing that verifies non-functional aspects of the product, such as performance, stability, and usability.

TYPES OF FUNCTIONAL TESTING

1. Unit Testing
2. Smoke Testing
3. Sanity Testing
4. Regression Testing
5. Acceptance Testing
6. Black Box Testing
7. White Box Testing
8. Grey Box Testing
9. Integration Testing

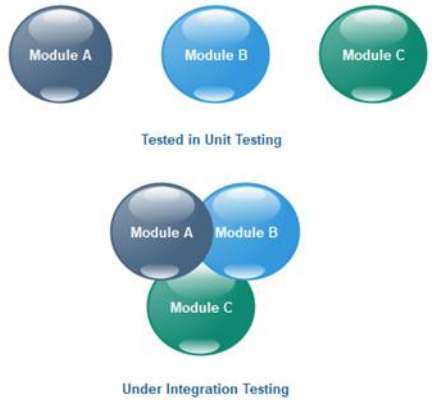
Unit testing is done at the code level, where each component is tested individually to ensure their impartiality and analyse their functionality.

Smoke Testing



Smoke testing, also called **build verification testing or confidence testing**, is a software testing method that is used to determine if a new software build is ready for the next testing phase. **This testing method determines, if the most crucial function of a program works, but does not delve into finer details**. Smoke testing test - Checking build is stable or not.

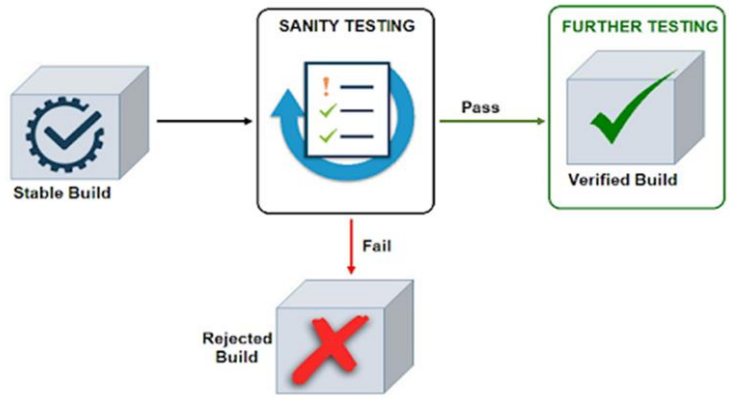
Integration testing is the second level of the software testing process comes after unit testing. In this testing, units or individual components of the software are tested in a group. The focus of the integration testing level is to expose defects at the time of interaction between integrated components or units.



Sanity Testing

Sanity testing is a type of software testing that aims **to quickly evaluate whether the basic functionality of a new software build is working correctly or not**. It is usually performed on builds that are in the initial stages of development before the full regression testing is performed.

It is also called subset of Regression Testing.



Regression Testing

Regression Testing is defined as a type of software testing to confirm that a recent program or code change has not adversely affected existing features. We can also say it is nothing but a full or partial selection of already executed test cases that are re-executed to ensure existing functionalities work fine.

Black Box testing

Black Box Testing is a software testing method in which the functionalities of software applications are tested without having knowledge of internal code structure, implementation details and internal paths. Black Box Testing mainly focuses on input and output of software applications, and it is entirely based on software requirements and specifications. It is also known as Behavioural Testing.

White Box Testing

White Box Testing is a testing technique in which software's internal structure, design, and coding are tested to verify input-output flow and improve design, usability, and security. In white box testing, code is visible to testers, so it is also called Clear box testing

Grey Box Testing

Grey Box Testing or Grey box testing is a software testing technique to test a software product or application with partial knowledge of internal structure of the application. The purpose of grey box testing is to search and identify the defects due to improper code structure or improper use of applications.

- In Black Box testing internal structure (code) is unknown
- In White Box testing internal structure (code) is known

• In Grey Box Testing internal structure (code) is partially known

UAT –User Acceptance Testing

Acceptance testing involves testing the system's Functional and Non-functional aspects, such as performance, security, usability, accessibility, compatibility, and reliability. Depending on the system's complexity. it contains alpha testing and beta testing

• **Alpha Testing**: This phase is conducted by internal testers, often within the organization, to validate the product's quality and functionality. It uses both white box and black box testing techniques to identify bugs and ensure the software is stable and working as intended. Alpha testing is performed in a controlled environment and can last for several weeks, with multiple test cycles.

• **Beta Testing**: This phase involves real users outside the organization who test the software in their own environments. Beta testing focuses on usability, functionality, security, and reliability, and it helps gather feedback that can be used to improve the product before its official release. Beta testing is typically shorter, lasting a few weeks, and it is unstructured, allowing users to use the software freely and provide feedback.

NON-FUNCTIONAL TESTING

1. Performance Testing
2. Load Testing
3. Stress Testing
4. Compatibility Testing
5. Usability Testing
6. Security Testing
7. Volume Testing
8. Penetration Testing
9. Migration Testing
10. Installation Testing

1. PERFORMANCE TESTING

Performance Testing is a testing measure that evaluates responsiveness stability of a computer, network, software program or device under a workload.

Performance testing is a non-functional software testing technique that determines how the stability, speed, scalability, and responsiveness of an application holds up under a given workload.

2.LOAD TESTING

Load Testing is a **non-functional software testing process in which the performance of software application is tested under a specific expected load. It determines how the software application behaves while being accessed by multiple users simultaneously.** The goal of Load Testing is to improve performance bottlenecks and to ensure stability and smooth functioning of software application before deployment.

3.STRESS TESTING

Stress Testing is a **type of software testing that verifies stability & reliability of software application. The goal of Stress testing is measuring software on its robustness and error handling capabilities under extremely heavy load conditions and ensuring that software doesn't crash under crunch situations.** It even tests beyond normal operating points and evaluates how software works under extreme conditions.

4.COMPATIBILITY TESTING

Compatibility Testing is a type of Software testing to check whether your software is capable of running on different hardware, operating systems, applications, network environments or Mobile devices.

5.USABILITY TESTING

Usability Testing also known as User Experience (UX) Testing, **is a testing method for measuring how easy and user friendly a software application is.** A small set of target end users, use software application to expose usability defects. Usability testing mainly focuses on user's ease of using application, flexibility of application to handle controls and ability of application to meet its objectives. **This testing is recommended during the initial design phase SDLC, which gives more visibility.**

6.SECURITY TESTING

The main goal of Security Testing is to **identify the threats in the system and measure the potential vulnerabilities,** so the threats can be encountered and the system does not stop functioning or cannot be exploited. **It also helps in checking all possible security risks in the system and helps developers to fix the problems through coding.**

7.PENETRATION TESTING

This kind of testing simulates an attack from a malicious hacker. This testing involves analysis of a particular system to check for potential vulnerabilities to an external hacking attempt.

8.VOLUME TESTING/FLOOD TESTING

Volume Testing is a type of Software Testing, where the software is subjected to a huge volume of data. It is also referred to as **Flood Testing**. **Volume testing is done to analyse the system performance by increasing the volume of data in the database. With the help of Volume testing, the impact on response time and system behaviour can be studied when exposed to a high volume of data.**

9.MIGRATION TESTING

Migration Testing is done to ensure that the software **can be moved from older system infrastructures to current system infrastructures without any issues**.

10.INSTALLATION TESTING

Installation testing is performed to **check if the software has been correctly installed with all the inherent features and that the product is working as per expectations.** Also known as **Implementation Testing**, it is done in the last phase of testing before the end-user has his/her first interaction with the product.

TESTING METHODS

1. Black Box Testing
2. White Box Testing
3. Grey Box Testing

1.BLACK BOX TESTING

Black Box Testing is a software testing method in which **the functionalities of software applications are tested without having knowledge of internal code structure, implementation details and internal paths.** Black Box Testing mainly focuses on input and output of software applications and it is entirely based on software requirements and specifications. It is also known as Behavioural Testing.

2.WHITE BOX TESTING

White Box Testing is a testing technique in which **software's internal structure, design, and coding are tested to verify input-output flow and improve design, usability, and security.** In white box testing, code is visible to testers, so it is also called Clear box testing.

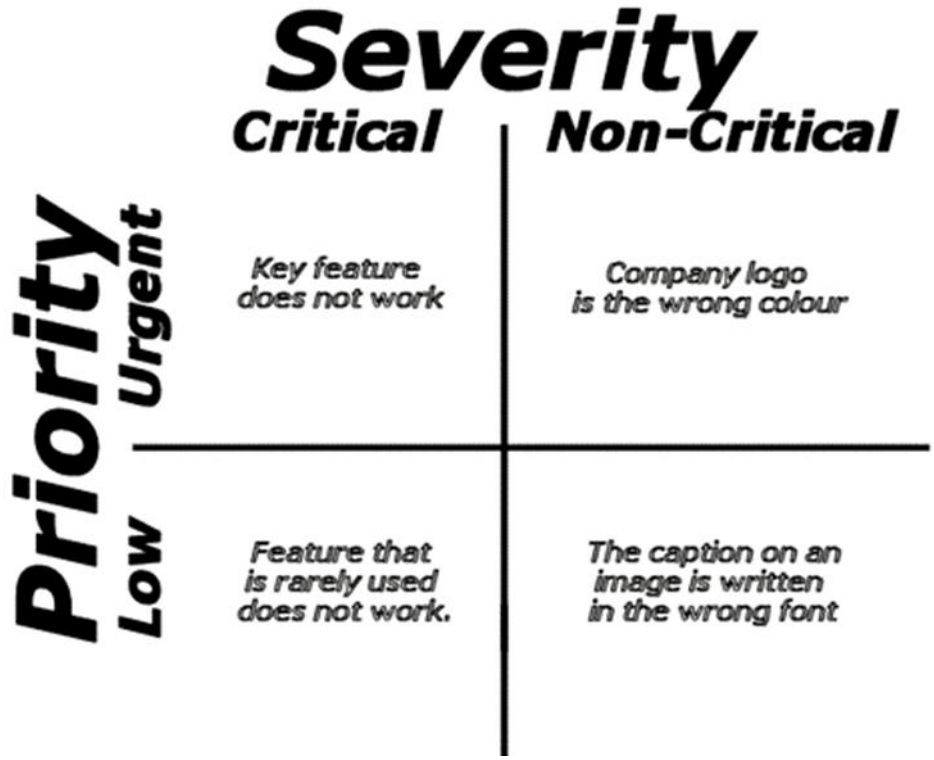
3.GREY BOX TESTING

Grey Box Testing or Gray box testing is a **software testing technique to test a software product or application with partial knowledge of internal structure of the application.** The purpose of grey box testing is to search and identify the defects due to improper code structure or improper use of applications.

- In Black Box testing internal structure (code) is unknown
- In White Box testing internal structure (code) is known

• In Grey Box Testing internal structure (code) is partially known

BUGS SEVERITY & PRIORITY



Bug Severity or Defect Severity in testing is a **degree of impact a bug or a Defect has on the software application under test.** A higher effect of bug/defect on system functionality will lead to a higher severity level.

Types of Severity of bug/defect can be categorized into the following parts:

• **Critical**: This defect indicates **complete shut-down of the process**, **nothing can proceed further**.

• **Major**: It is a **highly severe defect and collapses the system**. **However, certain parts of the system remain functional**.

• **Medium**: It causes some **undesirable behaviour, but the system is still functional.**

• **Minor/Cosmetic**: It **won't cause any major break-down of the system.**

PRIORITY

Priority is defined as the order in which a defect should be fixed. Higher the priority the sooner the defect should be resolved.

Types of Priority of bug/defect can be categorized into three parts:

• **High**: The defect must be resolved **as soon as possible as it affects the system severely and cannot be used until it is fixed**.

• **Medium**: **During the normal course of the development activities defect should be resolved.** It can wait until a new version is created.

• Low: The Defect is **an irritant, but repair can be done once the more serious Defect has been fixed**.

TEST DOCUMENTATION (These documents will be prepared during the scrum planning time)

1. Test Strategy
2. Test Plan
3. Test Approach
4. Test Report
5. QA Sign Off

1.TEST STRATEGY

Test Strategy Document is a well-described document in **software testing which clearly defines the exact software testing approach and testing objectives of the software application.** Test document is an important document for QA teams which is derived from actual business requirements that guides the whole team about software testing approach and objectives for each activity in the software testing process.

1. High level document given to client
2. Automation or manual testing
3. Number of team members
4. Test methodology

2.TEST PLAN

a. Scope:

The scope of a test defines what areas of a customer's product are supposed to get tested, what functionalities to focus on, what bug types the customer is interested in, and what areas or features should not be tested by any means

In Scope:

• In Scope defines the features, functional or non-functional requirements of the software that will be tested

Out of Scope:

• Out Of Scope defines the features, functional or non-functional requirements of the software that will NOT be tested.

b. **Objective**:

Test Objective is the overall goal and achievement of the test execution. The objective of the testing is finding as many software defects as possible; ensure that the software under test is bug free before release.

To define the test objectives, you should do 2 following steps :

1. List all the software features (functionality, performance, UI…) which may need to test.

2. Define the target or the goal of the test based on above features

c. Test Approach:

For the new user story testing, **the new functionality will be tested in manually and regression testing will be handled with automation.** The process model that is following will be an agile scrum that will be a monthly release.

Features to be Tested

Release ID	Epic ID	Epic Description	User Story ID	User Story Description
03.25.01	SCRUM-2	Contacts	SCRUM-18	Functionality to Add Customers
			SCRUM-15	Functionality to Add Suppliers
			SCRUM-12	Functionality of Suppliers Listing
			SCRUM-16	Functionality of Customers Listing

(*My Notes: Release ID from Jira – The version given under sprint 1.*

Epic – It's the Epic I do in the project

User Story ID – The ids of functionality

User Story Description – The name of functionalities)

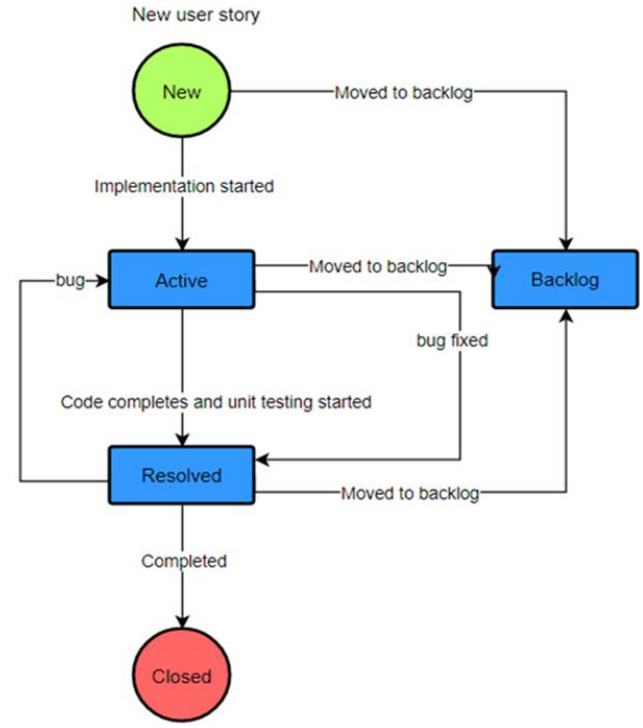
CONFIGURATIONS

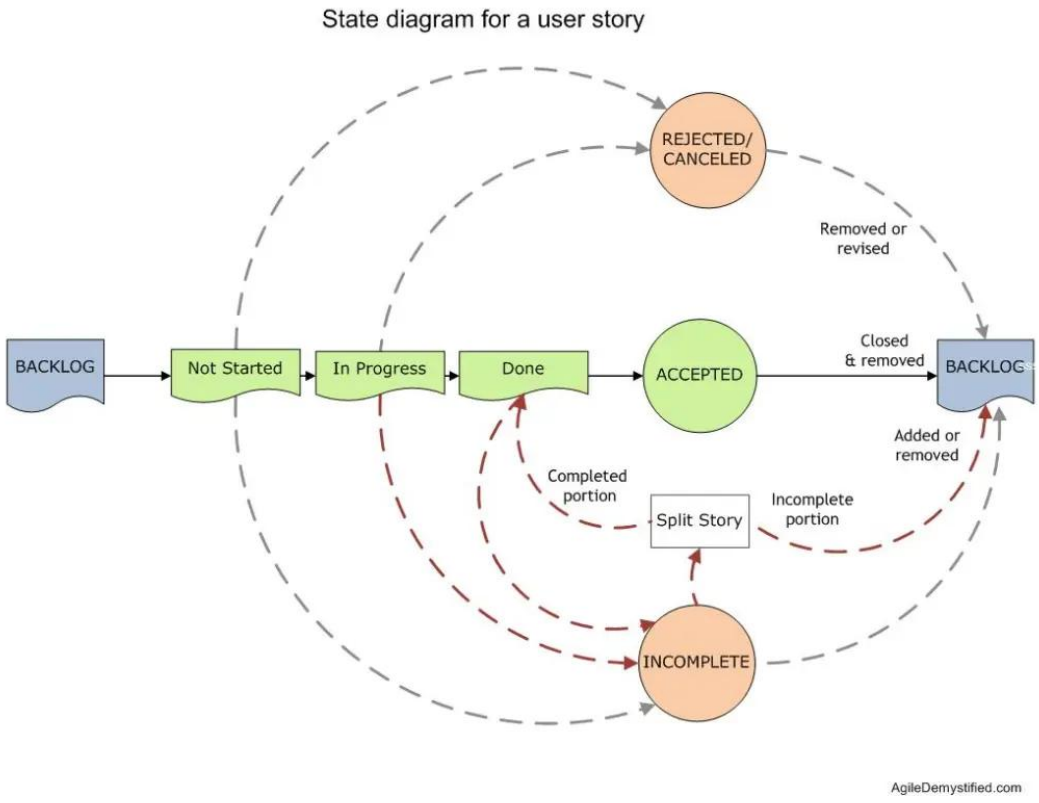
Release ID	Desktop Configuration	Device Configuration
03.25.01	Chrome	iPhone
04.25.02	Fire Fox	iPad
05.25.03	Safari	pixel
06.25.04	Edge	Galaxy

TEST MANAGEMENT

The test management will be handled with JIRA and the process model followed would be AGILE Scrum and the release would be in monthly basis.

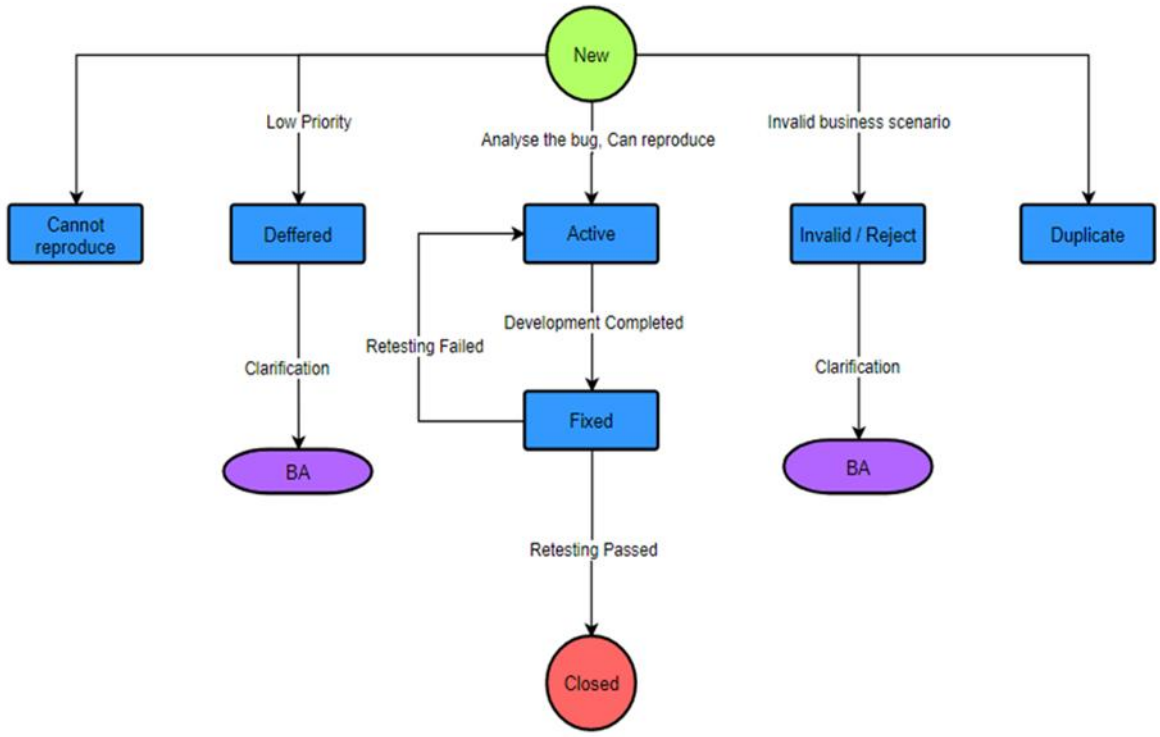
FLOW CHART OF USER STORY LIFECYCLE





Refer Notes - User Story & Bug Life Cycle

FLOW CHART OF BUG LIFECYCLE



Refer Notes - User Story & Bug Life Cycle

TEST SCHEDULE

User story testing will be starting in the first day of the sprint and will be end in the 16th day of the sprint. Regression testing will be handled for 2 days. User acceptance testing will be for 2 days. Product will be moved on 21th day of Sprint.

Activities	Start Date	End Date
New User Story Testing	26-03-2025	16-04-2025
Regression Testing	16-04-2025	18-04-2025
QA Sign Off	21-04-2025	22-04-2025
UAT Testing	22-04-2025	24-04-2025
Production Move	24-04-2025	25-04-2025

TEST DELIVERABLES

1. Test Plan Document
2. Test Cases
3. Test Report
4. Bug Report

RESOURCES

HUMAN RESOURCES

Resource Name	Designation	RACI(Responsible Accountable Consulted Informed)
Divya	QA Engineer	R
Saranya	Sr. QA Engineer	R
John	QA Lead	A
Sam	Dev Lead	A,C
Monica	Client Representative	I
Kin	QA Manager	C
Tom	Project Manager	I

SYSTEM RESOURCES

1.Testing Tools

- JIRA Test management tool is to enter and track all bugs and project issues.
- Test Case Management: Zephyr Squad
- Automation tools like Selenium, JMeter
- Configuration management: GIT
- Microsoft Excel, Microsoft Word
- Test Environment Dev, QA, Stage, UAT, Production

2.Execution Strategy

Entry Criteria

- All test cases should be reviewed and approved.
- Environment should be ready along with test data should be available.
- User story should be in resolved status.

Exit Criteria

- All test cases should be executed.
- There should not be any failed or blocked test cases.
- There should not be any active critical or blocker issue in open status.

• The release date should be met

RISK MANAGEMENT & MITIGATION

1.Resource Unavailability

2.Test data availability-User stories are tested in UAT environment

3.Environment dependency that can be – development environment, QI env, Production env, UIT env

3.TEST APPROACH

For the new user story testing, the new functionality will be tested in manually and regression testing will be handled with automation. The process model that is following will be an agile scrum that will be a monthly release.

4.TEST REPORTS

- Execution Report
- Bug Report

(*Check Execution and Bug Report - v1.xlxs*)

5.QA SIGN OFF

(*Refer QA Sign Off-v1.docx*)

TESTING QA QC

1. Quality
2. Quality Assurance
3. Quality Control
4. Verification
5. Validation
6. Bug, Error, Defect, Failure

1.QUALITY

Quality is extremely hard to define, and it is simply stated: "**Fit for use or purpose**." It is all about meeting the needs and expectations of customers with respect to functionality, design, reliability, durability, & price of the product.

2.QUALITY ASSURANCE

Quality Assurance in Software Testing is **defined as a procedure to ensure the quality of software products or services provided to the customers by an organization.** Quality assurance focuses on improving the software development process and making it efficient and effective as per the quality standards defined for software products. Quality Assurance is popularly known as QA Testing.

a. Process oriented

b. Prevention of defects

3.QUALITY CONTROL

Quality control popularly abbreviated as QC. It is a Software Engineering process used to ensure quality in a product or a service. **It does not deal with the processes used to create a product; rather it examines the quality of the "end products" and the final outcome.**

a. Product oriented.

b. Finding defects and resolving it.

4.VERIFICATION

Verification evaluates software artifacts (such as requirements, design, code, etc.) to ensure they meet the specified requirements and standards. It ensures the software is built according to the needs and design specifications.

Validation evaluates software to meet the user's needs and requirements. **It ensures the software fits its intended purpose and meets the user's expectations.**

5.ERROR, BUG, DEFECT, FAILURE

Error

An error is a mistake made by a developer in the code.

Its main causes are:

• Negligence: Carelessness often leads to errors

• Miscommunication: An unclear feature specification or its improper interpretation by developers could lead to slips.

• Inexperience: Inexperienced developers often miss out on essential details, leading to faults down the line.

• Complexity: Intricate algorithms can cause developers to make mistakes in their coding logic.

Bug

In software testing, a bug is a deviation from the customer requirement, in simple language, we can say **deviation between the expected result and the actual result in an application or in a module that is found by the testing team during the testing period.**

Defect

If the functionality of **an application is not working as per the customer's requirement is known as a defect. It is found during the development phase while unit testing.** Giving wrong input may lead to a defect or Any code error may lead to a defect.

Failure

Once the software is completed and delivered to the customer **if the customer finds any issue in the software, then it is the condition of failure of the software.** In other words, if an end user finds an issue in the software, then that particular issue is called a failure.

Reasons:

• Missing test cases

• Test cases not executed by the tester

• Improper execution of test cases

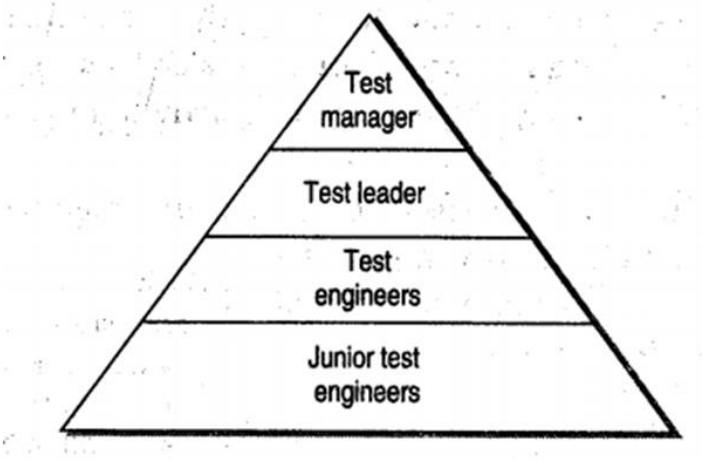
• Code changed after testing

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.

Req No	Req Desc	Testcase ID	Status
123	Login to the application	TC01,TC02,TC03	TC01-Pass TC02-Pass
345	Ticket Creation	TC04,TC05,TC06, TC07,TC08,TC09 TC010	TC04-Pass TC05-Pass TC06-Pass TC06-Fail TC07-No Run
456	Search Ticket	TC011,TC012, TC013,TC014	TC011-Pass TC012-Fail TC013-Pass TC014-No Run

QA TEAM STRUCTURE



Testing Group Hierarchy

TEST ENVIRONMENT SETUPS

- Entry Criteria
- Exit Criteria
- Test Bed
- Test Suite

Entry Criteria

- All test cases should be reviewed and approved.
- Environment should be ready along with test data should be available.
- User story should be in resolved status.

Exit Criteria

- All test cases should be executed.
- There should not be any failed or blocked test cases.
- There should not be any active critical or blocker issue in open status.
- The release date should be met

Test Bed

The testbed provides a realistic hardware-software environment with which to test components without having the ultimate system. The testbed provides a means to improve the understanding of the functional requirements and operational behaviour of the system.

Test Suite

Test suites are the logical grouping or collection of test cases to run a single job with different test scenarios. For instance, a test suite for product purchase has multiple test cases, like:

Test Case 1: Login.

Test Case 2: Adding Products.

Test Case 3: Checkout.

OTHER TESTING APPROACHES

1. Smoke & Sanity Testing
2. Regression Testing & Retesting
3. Adhoc Testing
4. Monkey Testing
5. Exploratory Testing

Smoke Testing

Smoke testing, also called **build verification testing or confidence testing**, is a software testing method that is used to determine if a new software build is ready for the next testing phase. **This testing method determines, if the most crucial function of a program works, but does not delve into finer details**. Smoke testing test - Checking build is stable or not. (Usually smoke testing can be done after unity testing. Or it is first).

Sanity Testing

Sanity testing is limited in scope and typically focuses on critical functionality and does not aim to uncover every possible error or bug in the system. It is a quick and lightweight way to ensure that the software is functioning as expected before further testing is conducted. Subset of Regression Testing. (It will be done after a detailed testing and the bugs fixed. It can be said as a quick evaluation)

Regression Testing & Retesting

Regression Testing is defined as a type of software testing to confirm that a recent program or code change has not adversely affected existing features. We can also say it is nothing but a full or partial selection of already executed test cases that are re-executed to ensure existing functionalities work fine.

Retesting is the process of testing a previously failed test case after its defects have been fixed. The retesting aims to confirm that the defects/bugs have been fixed and that the affected functionality works as expected.

Adhoc Testing

We know the functionality of the software, but we are not aware of the documentations, so that we will be doing a testing call as Ad hoc Testing.

Monkey Testing

It is a technique in software testing where the user tests the application by providing random inputs and check the behaviour or try to crash the application. Eg: Bank invalid password multiple times or more than 5 times.

Exploratory Testing

If you got an existing system for testing, the RTM (Requirement Traceability Matrix) and Test cases are missing. In this case, as a tester level, we need to understand the application and identify possible scenario ourselves.

STATIC & DYNAMIC TESTING

Static Testing

Static Testing also known as **Verification testing or Non-execution testing** is a **type of Software Testing method that is performed to check the defects in software without actually executing the code of the software application**.

1. Static testing is performed in the early stage of development to avoid errors as it is easier to find sources of failures and it can be fixed easily.

2. Static testing is performed in the white box testing phase of the software development where the programmer checks

3. The errors that can't not be found using Dynamic Testing, can be easily found by Static Testing.

4. It involves assessing the program code and documentation.

5. It **involves manual and automatic assessment of the software documents**.

Dynamic Testing

Dynamic Testing is a type of Software Testing that is performed to analyze the dynamic behavior of the code. It includes the testing of the software for the input values and output values that are analyzed.

1. **The purpose of dynamic testing is to confirm that the software product works in conformance with the business requirements.**

2. **It involves executing the software and validating the output with the expected outcome.**

3. It can be with black box testing or white box testing.

4. It is slightly complex as it requires the tester to have a deep knowledge of the system.

5. It provides more realistic results than static testing.

SOME ADDITIONAL NOTES

Product Backlog: A prioritized list of all desired work on the product, maintained by the Product Owner.

Sprint Backlog: A list of tasks selected from the Product Backlog that the team commits to complete during a specific sprint, along with a plan to deliver them.

Release Backlog: A subset of the Product Backlog that includes the items planned for a specific product release. It focuses on delivering a set of features or fixes by a certain date or milestone.

A **Spike Story** is a special type of user story in Agile used specifically for research, exploration, or prototyping when the team lacks sufficient information to estimate or implement a feature.

Zero Sprint: A preparatory phase before the first official sprint, used to set up the project environment, tools, and initial backlog.

Bug Release: Definition: A bug release occurs when a bug that was not identified during the testing phase is deliberately released into the production environment, often due to insufficient testing or oversight.

Bug Leakage: Definition: Bug leakage occurs when a bug that should have been detected during the testing phase slips through and is discovered by users or customers after the software has been released.

Positive Testing: Testing the application with valid and expected inputs to check if it behaves as intended.

Negative Testing: Testing the application with invalid, unexpected, or incorrect inputs to ensure it handles errors gracefully.

A **Test Driver** is a piece of code or program used to control and invoke the unit or module being tested. It is often used to simulate the behavior of higher-level modules that interact with the module being tested.

Test Stub: A Test Stub is a piece of code or a dummy implementation of a module or function that is used to simulate the behaviour of a component that is not yet available or is complex to test.