

# **Appendix 1**

## **Module Title: Software Testing Overview**

**Credits: 04**

### **Module Description**

Software testing is an art of gathering information through manual or automation means by making observation and comparing with actual requirement. Testing a software or a piece of code is an integral part of software development life cycle. Software testing taking place in all most all the phases, starting requirement to implementation and followed by maintenance of a software. This complete course will walk you through all the concepts & mythologies of software Testing.

### **Module Introduction**

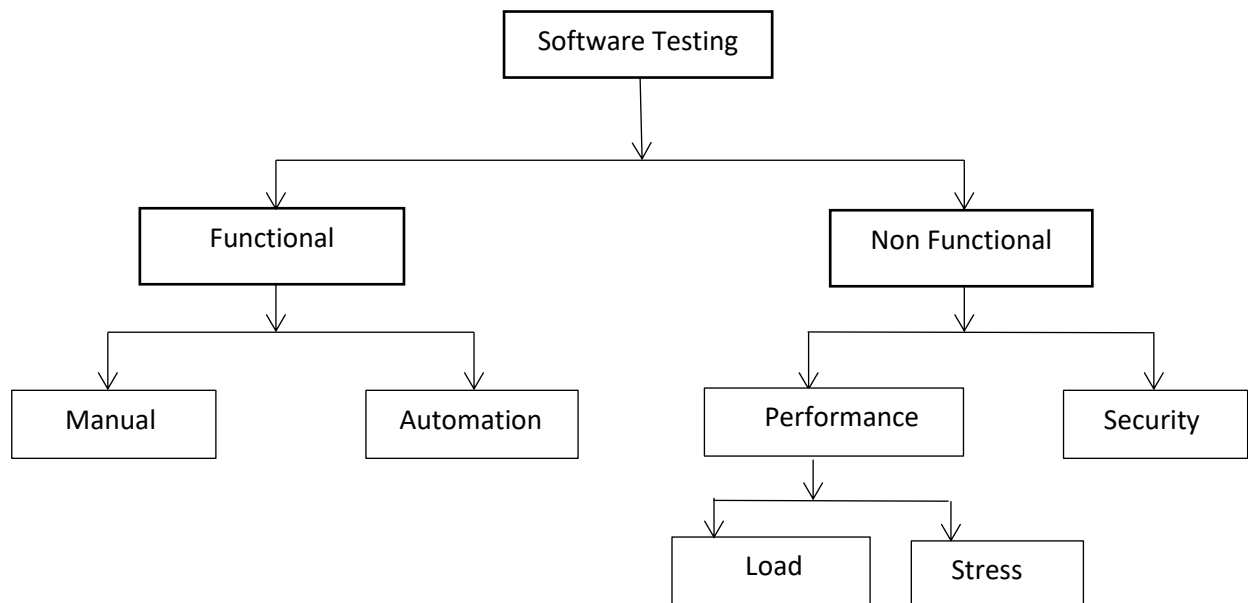
The course materials are so designed as to acquire theoretical and practical knowledge in software testing, Development life cycle, its types and other related terminologies. The course is divided into 5 modules. All modules have equal importance. Each module is divided into units in a sequential order. Since it is written in a simple language a learner can easily understand it.

### **Module Objectives**

On successful completion of this course, it is hoped that you will be able to:

- Understand what Software Testing is & why Software Testing..!
- Understanding about Software Development Life Cycle.
- Learn how to design, develop test scenarios and test cases.
- Understanding about the different types of software testing.

## ❖ Software Testing-Big Picture



### 1.1 Beginner Guide to Software Testing

- 1.1.1 Requirement Analysis
- 1.1.2 Use Case
- 1.1.3 Business Requirement Document(BRD)
- 1.1.4 System Requirement Specification(SRS)
- 1.1.5 Review and baseline Process, Baseline Requirement, Change Request Management Process, Baseline Document.
- 1.1.6 User Story(Summary, Description, DOD, Acceptance Criteria)
- 1.1.7 Testing, Software Testing

On completion of this unit, the learner will be able to have an understanding on how a software will evolve from a piece of requirement and other related terminologies.

Lab: Candidates Needs to Create requirements document for 4 user cases.

## 1.2 Software Development Life Cycle

- 1.2.1 Importance of SDLC
- 1.2.2 SDLC Phases
- 1.2.3 Types of SDLC Model

On completion of this unit, the learner will be able to understand the different Software development approaches.

## 1.3 Software Testing Life Cycle

- 1.3.1 STLC Phases

## 1.4 Alphabets of Manual Testing

- 1.4.1 Test Scenario
- 1.4.2 Test Case preparation, Template
- 1.4.3 Test Case design techniques, Test Data
- 1.4.4 Bug, Error, Defect, Failure
- 1.4.5 Testing tasks & Activities(Coding, Code Review, Unit Testing, Test Case Design, Test Case Review, Test Data Preparation/Configuration, Test Execution)

On completion of this unit, the learner will be able to design and develop test cases to specific requirements.

Lab: Develop & Review Test Cases for 10 User Stories

## 1.5 Types of Testing

- 1.5.1 Functional Testing
- 1.5.2 Non Functional Testing
- 1.5.3 Automation Testing

On completion of this unit, the learner will be able to understand the different types of software testing.

## Appendix 2

### Module Title: Software Testing Complete Guide

#### Module Description

Now we will see some **more details about web application testing with web testing test cases**. I always love to share practical knowledge, which in case can be useful to several users in their career life. The course is specially designed for beginners with little or no Testing experience. This complete course will walk you through all the concepts & mythologies of software Testing.

#### Module Introduction

The course materials are so designed as to acquire theoretical and practical knowledge in software testing, QA, QC, various testing methods other related concepts. The course is divided into 10 modules. All modules have equal importance. Each module is divided into units in a sequential order. Since it is written in a simple language a learner can easily understand it.

#### Module Objectives

On successful completion of this course, it is hoped that you will be able to:

- Understand What is QA and what QC is.
- Understanding about different Software Testing Methods.
- Learn how to design, develop Test artifacts.
- Understanding about the different levels of software testing and Agile SDLC Model.

#### 2.1 Testing Myths

- 2.1.1 Testing is too expensive and Time consuming
- 2.1.2 Only fully developed products are tested
- 2.1.3 Complete testing is possible
- 2.1.4 Bug free software
- 2.1.5 Missed defects
- 2.1.6 Testing Principles

On completion of this unit, the learner will be able to understand the mythologies in software testing.

## 2.2 Test Design Techniques

## 2.3 Test Approaches

- 2.3.1 Smoke Testing
- 2.3.2 Sanity Testing
- 2.3.3 Retesting
- 2.3.4 Regression Testing
- 2.3.5 Ad-hoc Testing
- 2.3.6 Monkey Testing
- 2.3.7 Exploratory Testing
- 2.3.8 Compatibility Testing

On completion of this unit, the learner will be able to understand different Black box test design techniques.

Lab: Develop test cases for 10 User stories and complete peer review.

## 2.4 Testing – QA & QC

- 2.4.1 Verification & Validation
- 2.4.2 Quality Assurance & Quality Control
- 2.4.3 Bug, Error, Defect, Failure

On completion of this unit, the learner will be able to understand the process, procedures and best practices in software testing.

## 2.5 Testing Methods

- 2.5.1 Black Box Testing
- 2.5.2 White Box Testing
- 2.5.3 Grey Box Testing

On completion of this unit, the learner will be able to understand the different software Testing methodologies.

## 2.6 Levels of Testing

- 2.6.1 Unit Testing
- 2.6.2 Integration Testing
- 2.6.3 System Testing
- 2.6.4 Acceptance Testing, Alpha – Beta Testing

On completion of this unit, the learner will be able to understand the level of software Testing.

## 2.7 Documentation

- 2.7.1 Test Strategy
- 2.7.2 Test Plan
- 2.7.3 Test Approach
- 2.7.4 Test Reports
- 2.7.5 QA Sign off

## 2.8 Agile Development Model

- 2.8.1 Scrum
- 2.8.2 Product Backlog
- 2.8.3 Scrum Practices
- 2.8.4 Scrum Meetings
- 2.8.5 Story Points

On completion of this unit, the learner will be able to understand the various test artifacts in software Testing.

Lab: Candidates need to create all the above documents and reviewed and signoff by the trainer.

## 2.9 Test Management

- 2.9.1 Bug Life Cycle(Refer our Utube channel)
- 2.9.2 Bug Reporting & Tracking
- 2.9.3 Jira
- 2.9.4 Severity(Critical, High, Medium, Low) & Priority(High, Medium, Low)
- 2.9.5 Test Coverage Management - RTM
- 2.9.6 Metrics & Measures
- 2.9.7 QA Team Structure , Different Roles and Responsibilities

On completion of this unit, the learner will get a better understanding about Functional flow of Testing, Defect Management and overall functioning of a QA Team.

## 2.10 Other Test Approaches

- 2.10.1 Smoke – Sanity
- 2.10.2 Retesting – Regression testing
- 2.10.3 Adhoc Testing
- 2.10.4 Monkey Testing
- 2.10.5 Exploratory Testing
- 2.10.6 Compatibility Testing

On completion of this unit, the learner will be able to understand different test approaches.

## **Appendix 3**

### **Module Title: Test Metrics/Measures and Estimation Detail Study.**

#### **Module Description**

Measuring and evaluating the testing effort is always important an important aspect. All the testing efforts are expected to be thoroughly evaluated before assigning to the testers. The estimations should come to user story level from starting from Epic, Features etc. Test estimation taking place in all most all the phases of Software Test life cycle. This complete course will detail you through all the Estimation techniques, Metrics and measures.

#### **Module Introduction**

The course materials are so designed as to acquire theoretical and practical knowledge in software Test estimation, Metrics, Root Cause Analysis, QA Team Structure and other related concepts. The course is divided into 4 modules. All modules have equal importance. Each module is divided into units in a sequential order. Since it is written in a simple language a learner can easily understand it.

#### **Module Objectives**

On successful completion of this course, it is hoped that you will be able to:

- Understand What is Test Estimation and why.
- Understanding about different Software Test estimation Methods.
- Learn about test Metrics and Measures
- Understanding about QA Organization.

#### **3.1 Test Estimation**

- 3.1.1 Work Break Down Structure
- 3.1.2 Three Point Estimation
- 3.1.3 Delphi Technique
- 3.1.4 Test Case Point Estimation.

On completion of this unit, the learner will be able to understand the various Test estimation techniques.

## 3.2 Test Environment Setup

- 3.2.1 Entry Criteria
- 3.2.2 Exit Criteria
- 3.2.3 Test Bed
- 3.2.4 Test Suite

On completion of this unit, the learner will be able to setup a test environment to test.

## 3.3 Root Cause analysis

- 3.3.1 Why Root Case analysis
- 3.3.2 Fish Bone Analysis
- 3.3.3 5 Whys technique

On completion of this unit, the learner will be able to perform Root case analysis.

## 3.4 Test Metrics, QA Team

- 3.4.1 Subject to change to organizations.
- 3.4.2 Resource utilization
- 3.4.3 Productivity
- 3.4.4 Capacity Planning
- 3.4.5 QA Team Structure
- 3.4.6 Different Roles and Responsibilities

On completion of this unit, the learner will be able understand Test Metrics, resource utilization, productivity and capacity of the team and over all QA Team structure.



# **MODULE - 1**

## **SOFTWARE TESTING OVERVIEW**

### **1.1 A BEGINNER GUIDE TO SOFTWARE TESTING**

#### **1.1.1 Requirement Analysis**

Requirement is an idea or expectation or imagination about the business which I really want to run in that software. Through this requirement only the client can run this business.

Requirements can be collected in multiple ways:

- One to one interaction with clients.
- Skype call
- Face to face conversation.

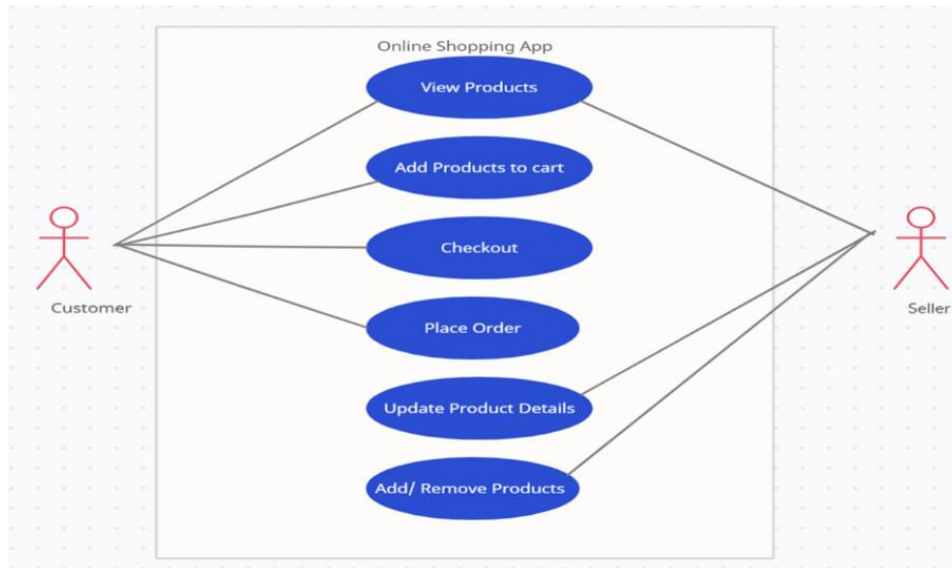
**E.g :** Requirement of pen

- The pen must write smoothly on paper or other surfaces.
- Consistent ink flow without blotting or skipping.
- It should be a ball point pen.
- Ink should be refillable.
- Comfortable grip for extended writing periods .
- Cap should be removable.
- The pen should withstand regular use without breaking or wearing out quickly.
- Ink should be blue color.
- Should be fine tip size.
- Ink should be erasable.

#### **1.1.2 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 perspective you can write a use case for various purposes such as testing a software or creating a guide manual for customers.

Use case are the first form of collecting requirements



### Example:

E-Commerce Website

### Use case 1

#### User Registration

Actor- new user

#### Main flow

1. User clicks on signup button
2. User enters personal input name, email, password
3. System validates information and ensure email is unique and password meets security criteria
4. User submit registration
5. System sends verification mail
6. User verifies email and access to the E-commerce page

### 1.1.3 Business Requirement Document (BRD):

A business requirements document (BRD), is a formal report that details all the objectives or “requirements” for a new project, program or business solution. The Business Requirement Document (BRD) is written for clients from a business perspective. The target audience of the BRD Document is business users. It is a high-level document.

### Business Analyst (BA)

Business analyst is a person who has technical knowledge as well as business knowledge.

**E.g.:** A 5 year or 7-year experienced QA person predominantly working on an E-Commerce domain means he will be having greater understanding in the E-Commerce domain. He can act as a BA. Because he knows business.

BA is responsible for getting the requirement finalized and drafted in the requirement docs based on which product owner creates the user story.

If there are any ambiguities in user stories/acceptance criteria he is the one who is approached by the scrum team. He then takes it to the PO. Scrum master acts as BA in small projects. Development team involves coders and testers.

#### **1.1.4 System Requirement Specification (SRS) :**

Software Requirement Specification document

The technical conversion of the BRD document is called the Software Requirements Specification (SRS) document. Main target of the SRS document is the technical team. Technical team includes both coders and testers. SRS includes everything which should be designed and developed during the project life cycle.

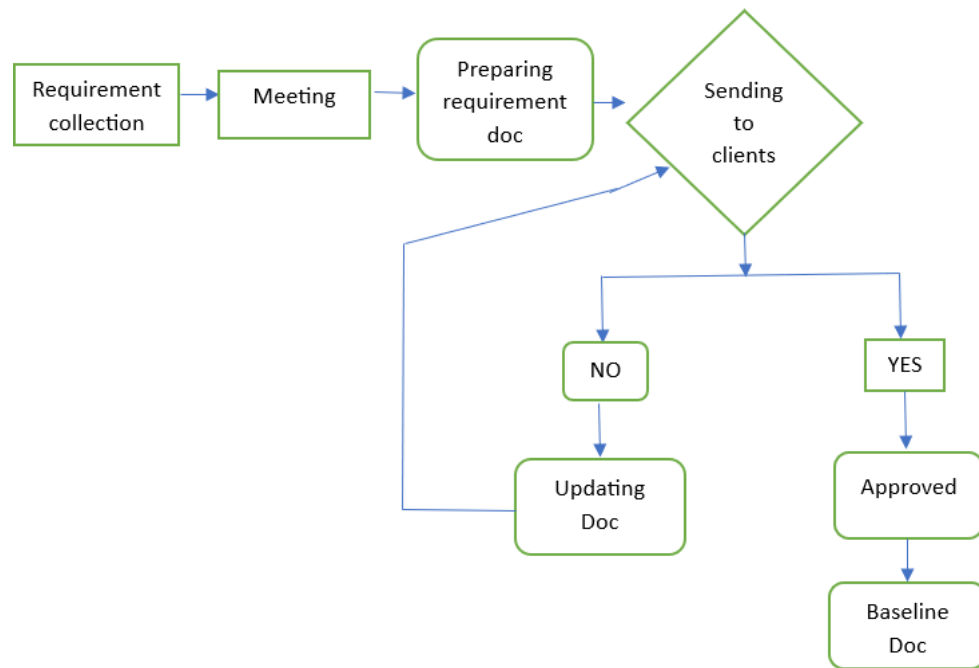
#### **1.1.5 Change Request Management and Baseline Requirement Document**

A **baseline** requirements document is a set of requirements that have been agreed upon and **approved** for a specific product release. It can also be called documentation of certain requirement items, attributes, and artifacts at any given point in time.

A change request is a formal proposal to modify some aspect of a project, system, process, or document. Change requests are typically submitted when there's a need to alter the scope, schedule, budget, or any other aspect of a project that has already been approved or initiated.

E g: A change request was submitted by the client, ABC Corp, to add a "Wishlist" feature to their online shopping website. This feature allows users to save products for future reference, enhancing user experience. The change would require an additional 2 weeks for development and

1 week for testing, increasing the project cost by \$7,000. The project steering committee approved the request, recognizing the impact on the project timeline and budget. The development was scheduled to begin on 25th October 2024 and be completed by 15th November 2024. After thorough testing, the feature was successfully deployed with no major issues, enhancing the website's functionality.



### 1.1.6 User Story

How a user is going to define the requirement. They are the requirements or features which have to be implemented. In scrum, requirements are defined in a single para as we don't have huge requirement documents.

The following format is used

**As a user**

**I want to <some achievable goal / target>**

**So that I <some results or reason for doing the thing>**

**E g: As a user, I want to** select a product **So that** I can buy it later.

It is the responsibility of the scrum master and BA to make sure that the PO has drafted the user stories correctly with a proper set of acceptance criteria.

Acceptance criteria: Its further details down the user story.

**Contents of user story-**

**a. Summary** – E g: [RESTAURANT][ ADD PRODUCT]

**b.DOD-** Definition of done.

Syntax - As a user,

I want to <some achievable goal / target>

So that I <some results or reason for doing the thing>

**Eg:** As a user I want to add a product to the list so that I can view the product in the products list.

### **c. Acceptance Criteria**

**Eg:** I want to sort by suppliers list. It should be given on the top right

I want to sort by product type. It should be given on the top right

I want to sort by number of entries. It should be given on the top left.

**Summary :** [Rainbow][grocery]: Update My Profile

**DoD:** As a user , I want to access my Account page, So that I can update/View My personal details and communication details

AC:

1. Only registered user should be able to access My account page
2. My Profile UI design document has been attached for reference:
3. All the fields are mandatory
4. Validation Message:
  - a) if the fields are blank, Name - Please enter Name,  
Email -Please enter Email  
Telephone- Please enter Telephone
  - b) If the fields contains special char, Name - Invalid Name  
Email - Invalid Email  
Telephone- Invalid Telephone
5. If Image blank - Please upload Image  
Invalid Image Format - Invalid Image format, accepts only jpg format.
6. On successful Update message should be displayed as:  
Alert! Personal Details Updated Successfully

### **1.1.7 Testing**

Testing is executing the program with the intention of finding errors. E.g.: Google pay

Testing is the process of evaluating a system or its components to determine whether it meets specified requirements and functions correctly. It involves identifying and fixing defects to ensure the quality, functionality, reliability, and performance of the product.

## Software Testing

Software Testing is an art of gathering information through manual or automation means by making observations and comparing with actual requirements. It involves execution of software/system components using manual or automated tools to evaluate one or more properties of interest. The purpose of software testing is to identify errors, gaps or missing requirements in contrast to actual requirements.

### Need of Software Testing

- Assurance to customer product works as intended
- Saves time of developer by detecting defects in early stage
- Saves reputation of company

### ❖ Characteristics of Tester

- Ensure end user satisfaction
- Think from user perspective
- Be a good observer
- Develop good analyzing skills
- Learn to negotiate with developers

### ❖ Benefits of Software Testing

- **Cost-Effective:** It is one of the important advantages of software testing. Testing any IT project on time helps you to save your money for the long term. In case if the bugs caught in the earlier stage of software testing, it costs less to fix.
- **Security:** It is the most vulnerable and sensitive benefit of software testing. People are looking for trusted products. It helps in removing risks and problems earlier.
- **Product quality:** It is an essential requirement of any software product. Testing ensures a quality product is delivered to customers.
- **Customer Satisfaction:** The main aim of any product is to give satisfaction to their customers. UI/UX Testing ensures the best user experience.

### ❖ Types of Organization

In IT Industry, the companies can be categorized into 2 ways

#### 1. Service Based Company:

A service-based company provides services rather than products, focusing on fulfilling client needs. In IT, these companies offer services like software development, consulting, support, and outsourcing. They work on a contract basis, often handling projects for various clients. Examples include TCS, Infosys, and Accenture, and their offerings include software development, IT consulting, quality assurance, and business process outsourcing (BPO).

## 2. Product Based Company:

Product-based company creates and sells its own products, often focusing on software, hardware, or digital platforms. These companies develop products that are used by many customers and continuously improve them based on market demand. They own the product lifecycle from design to maintenance and updates. Examples include Google, Microsoft, and Adobe, which sell software products like Google Workspace, Windows, and Adobe Photoshop. Unlike service-based companies, their revenue comes from selling or licensing their products directly to users or businesses.

### Types of Applications:

- **Desktop Application:** Software installed and run locally on a computer (Windows, Mac, Linux). Examples: Microsoft Word, Adobe Photoshop.
- **Web Application:** Software accessed through a web browser over the internet. It's platform-independent and doesn't require installation. Examples: Gmail, Google Docs.
- **Mobile Application:** Software designed for smartphones or tablets, installed via app stores (iOS, Android). Examples: WhatsApp, Instagram.

## **1.2 SDLC – SOFTWARE DEVELOPMENT LIFE CYCLE**

**SDLC** is a systematic process for building software that ensures the quality and correctness of the software built. SDLC process aims to produce high-quality software that meets customer expectations. The system development should be complete in the pre-defined time frame and cost. SDLC consists of a detailed plan which explains how to plan, build, and maintain specific software. Every phase of the SDLC life Cycle has its own process and deliverables that feed into the next phase.

SDLC stands for **Software Development Life Cycle** and is also referred to as the Application Development life-cycle.

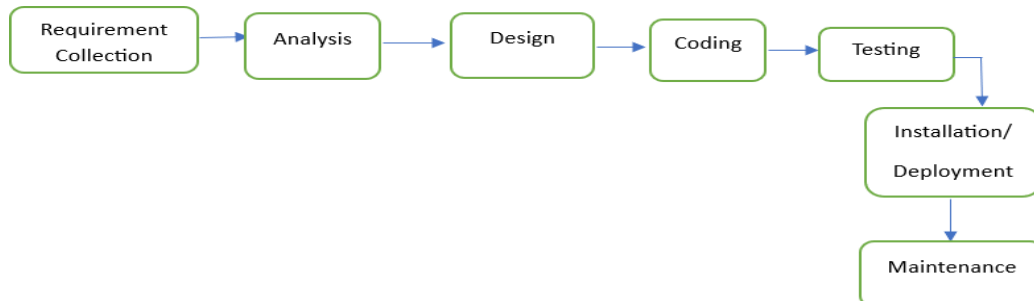
### **1.2.1 Importance of SDLC**

- Provides a framework for a standard set of activities and deliverables
- It is a mechanism for project tracking and control
- Increases visibility of project planning to all involved stakeholders of the development process
- Increased and enhance development speed
- Improved client relations
- Helps you to decrease project risk and project management plan overhead



### 1.2.2 SDLC Phases:

The entire SDLC process divided into the following SDLC steps: Requirement collection, Analysis, Design, Coding, Testing , Deployment , Maintenance.



#### ❖ Phase 1: Requirement collection

The requirement is the first stage in the SDLC process. In this phase requirements are gathered from clients. This stage gives a clearer picture of the scope of the entire project and the anticipated issues, opportunities, and directives which triggered the project. This helps companies to finalize the necessary timeline to finish the work of that system. Senior team members, domain experts, stakeholders will be part of these phase. The out put of this phase will be a high level BRD Document (Business Requirement Specification).

#### ❖ Phase 2: Analysis

Once the requirement analysis phase is completed the next sdlc step is to define and document software needs. It checks the feasibility requirement.

Feasibility:

- Economic feasibility
- Legal feasibility
- Technical feasibility
- Schedule feasibility
- Operations feasibility

In this phase the Software Requirement Specification document is created. SRS includes everything which should be designed and developed during the project life cycle.

#### Feasibilities checks:

- **Economic:** Can we complete the project within the budget or not?
- **Legal:** Can we handle this project as cyber law and other regulatory framework/compliances.
- **Operation feasibility:** Can we create operations which is expected by the client?
- **Technical:** Need to check whether the current computer system can support the software
- **Schedule:** Decide that the project can be completed within the given schedule ornot.

### ❖ Phase 3: Design

In this phase, the system and software design documents are prepared as per the requirement specification document. This helps define overall system architecture. This design phase serves as input for the next phase of the model. Two kinds of design documents developed in this phase:

#### High-Level Design (HLD)

- Brief description and name of each module
- An outline about the functionality of every module
- Interface relationship and dependencies between modules
- Database tables identified along with their key elements
- Complete architecture dialogues along with technology details

#### Low-Level Design (LLD)

- Functional logic of the modules
- Database tables, which include type and size
- Complete detail of the interface
- Addresses all types of dependency issues
- Listing of error messages
- Complete input and outputs for every module

### ❖ Phase 4: Coding

Once the system design phase is over, the next phase is coding. In this phase, developers start to build the entire system by writing code using the chosen programming language. In the coding phase, tasks are divided into units or modules and assigned to the various developers. It is the longest phase of the Software Development Life Cycle process.

### ❖ Phase 5: Testing

Once the software is complete, it is deployed in the testing environment. The testing team starts testing the functionality of the entire system. This is done to verify that the entire application works according to the customer requirement. During this phase, the QA and testing team may find some bugs/defects which they communicate to developers. The development team fixes the bug and sends it back to QA for a re-test. This process continues until the software is bug-free, stable, and working according to the business needs of that system.

#### ❖ **Phase 6: Installation/Deployment**

Once the software testing phase is over and no bugs or errors left in the system then the final deployment process starts. Based on the feedback given by the project manager, the final software is released and checked for deployment issues if any.

#### ❖ **Phase 7: Maintenance**

Once the system is deployed, and customers start using the developed system, following 3 activities occur.

- Bug fixing: Bugs are reported because of some scenarios which are not tested at all
- Upgrade: Upgrading the application to the newer versions of the Software
- Enhancement: Adding some new features into the existing software

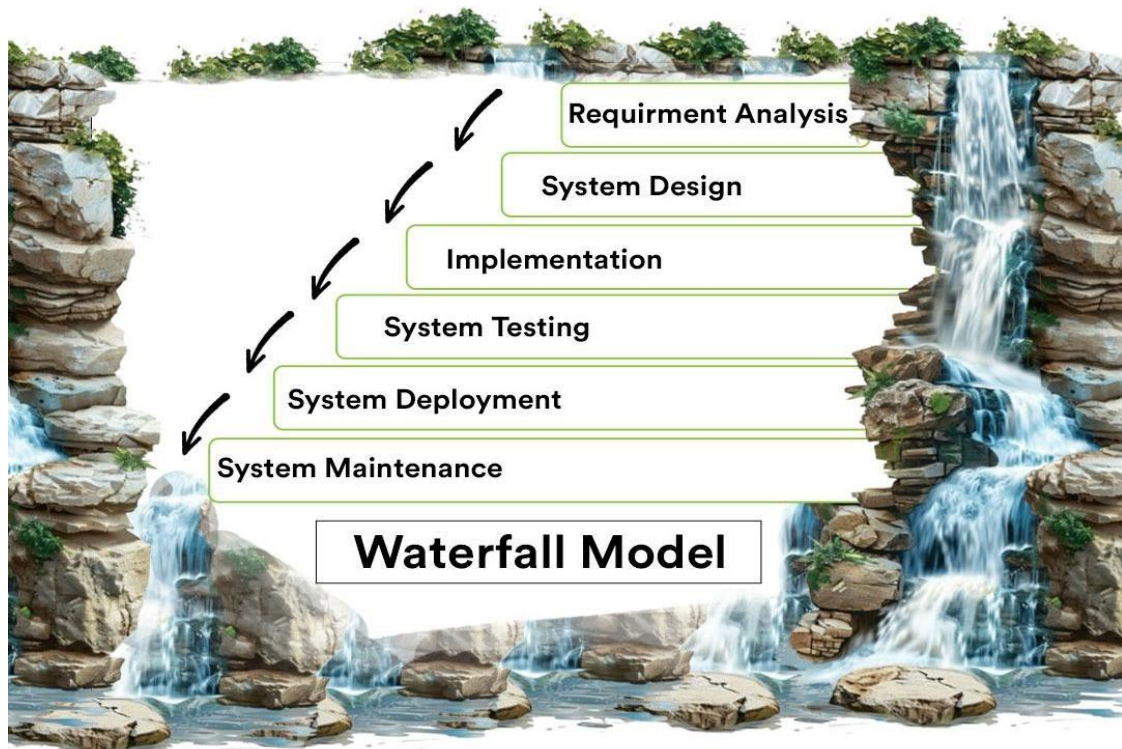
### **1.2.3 Types of SDLC Model**

#### **a) Waterfall model in SDLC**

- **Waterfall Model** is a sequential model that divides software development into pre-defined phases.
- Each phase must be completed before the next phase can begin with no overlap between the phases.
- The outcome of one phase is the input for next phase.
- Each phase is designed for performing specific activity during the SDLC phase.
- Top-down approach.
- It can be used only if the requirement is fixed.
- It was introduced in 1970 by Winston Royce.
- Application is not complicated and big
- Project is short
- Requirement is clear
- Environment is stable
- Technology and tools used are not dynamic and is stable

## Advantages and Disadvantages of Waterfall Model

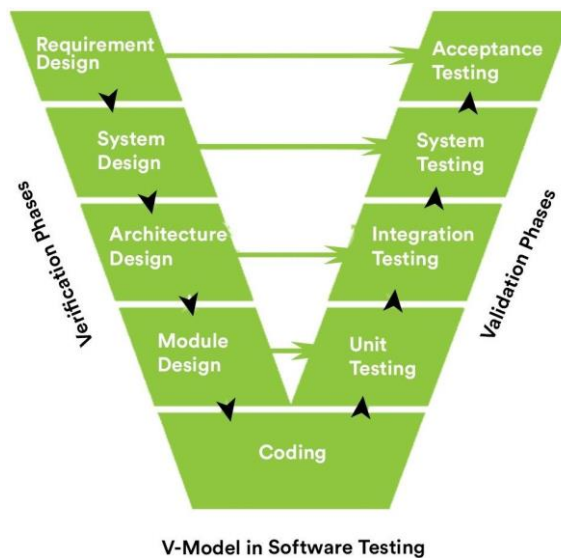
Advantages	Disadvantages
Suited for small projects where requirements are well defined	Not desirable for complex projects where requirements change frequently
They should perform quality assurance test (Verification and Validation) before completing each stage	Testing period comes quite late in the developmental process
Any changes in software is made during the process of the development	Small changes or errors that arise in the completed software may cause a lot of problems



## WATERFALL MODEL

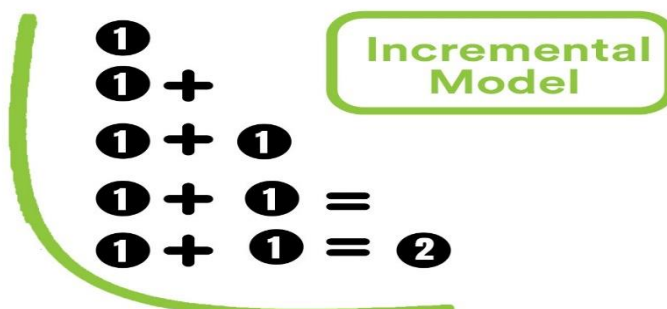
### b) V-Model in SDLC

- **V Model** is a highly disciplined SDLC model which has a testing phase parallel to each development phase.
- It is known as the Verification- Validation Model.
- LHS represents SDLC and RHS represents STLC.
- Testing early means verification of reviews to find defects.
- Each development level will have a test plan that defines the expected result, entry and exit criteria.

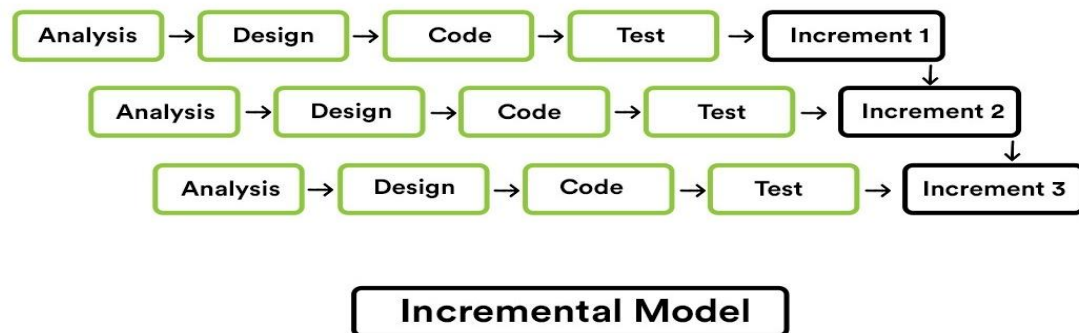


### c) Incremental Model in SDLC

Incremental Model is a process of software development where requirements are broken down into multiple standalone modules of the 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.



The system is put into production when the first increment is delivered. The first increment is often a core product where the basic requirements are addressed, and supplementary features are added in the next increments. Once the core product is analyzed by the client, there is plan development for the next increment.

#### Characteristics of an Incremental module includes

- System development is broken down into many mini development projects.
- Partial systems are successively built to produce a final total system.
- Highest priority requirement is tackled first.
- Once the requirement is developed, requirement for that increment is frozen.

Incremental Phases	Activities performed
Requirement analysis	Requirement and specification of the software are collected
Design	Some high-end functions are designed during this stage
Code	Coding of software is done during this stage
Test	Once the system is deployed, it goes through the testing phase.

#### When to use Incremental models?

- Requirements of the system are clearly understood.
- When demand for an early release of a product arises.

- When the team is not very well skilled or trained.
- When high-risk features and goals are involved.
- Such methodology is more in use for web application and product-based companies.

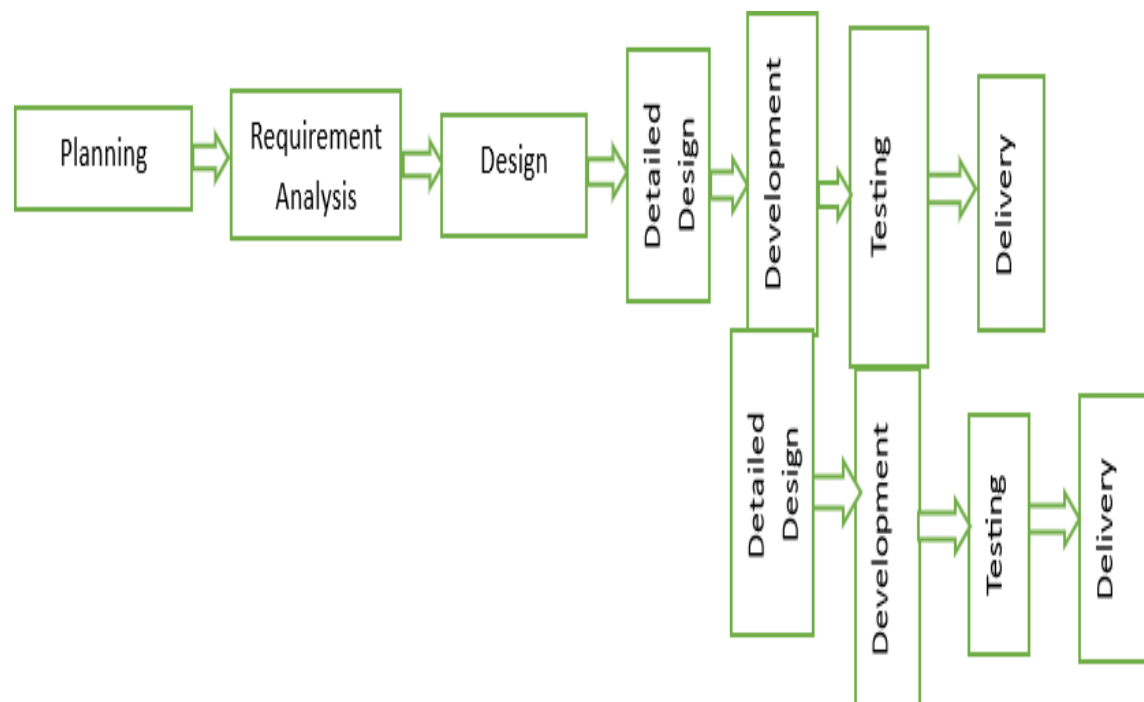
Advantages	Disadvantages
The software will be generated quickly during the software life cycle	It requires a good planning designing
Throughout the development stages changes can be done	Each iteration phase is rigid and does not overlap each other
This model is less costly compared to others	Rectifying a problem in one unit requires correction in all the units and consumes a lot of time

#### d) Iterative model

An iterative lifecycle model does not attempt to start with a full specification of requirements.

Instead, development begins by specifying and implementing just part of the software, which can then be reviewed in order to identify further requirements.

This process is then repeated producing a new version of the software for each cycle of the model.





As shown in the image above, when we work iteratively, we create a rough product, or product piece in one iteration, and then we review it and improve it in the next iteration and so on until it's finished.

#### ❖ **Advantages of Iterative Model:**

1. Testing and debugging during smaller iterations is easy.
2. A Parallel development can be planned.
3. It is easily acceptable to the ever-changing needs of the project.
4. Risks are identified and resolved during iteration.
5. Limited time spent on documentation and extra time on designing.

#### ❖ **Disadvantages of Iterative Model:**

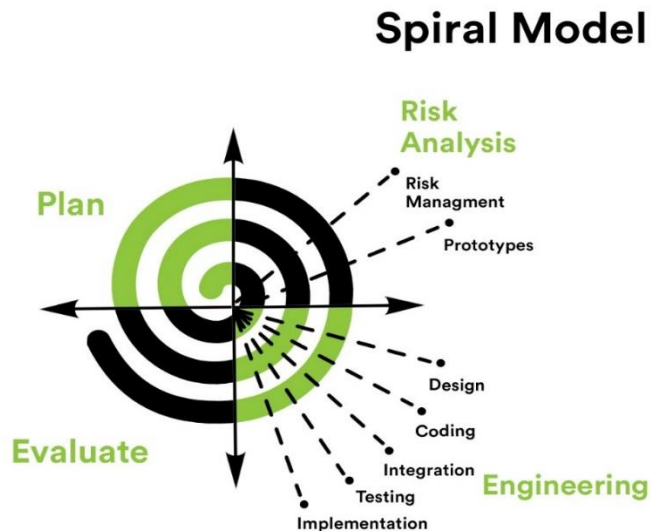
1. It is not suitable for smaller projects.
2. More Resources may be required.
3. Design can be changed again and again because of imperfect requirements.
4. Requirement changes can cause over budget.
5. Project completion date not confirmed because of changing requirements.

<b>Iterative Model</b>	<b>Incremental Model</b>
Develop through repeated cycles (iterations).	Build and deliver in pieces (increments).
Refine and revisit the same parts of the software.	Add new portions or features in each increment.
Focus on improving, polishing, and correcting.	Start with a core set of features, adding more over time.
Gradual refinement of the same components.	Each increment adds new functionality.
A final product after multiple refinements.	A complete system built incrementally.
Feedback refines existing components across cycles.	Feedback informs future increments but adds new features.
Useful when a product needs revisions.	Useful for delivering products in stages with defined features.



### e) Spiral Model

The Spiral Model is a combination of the waterfall model and the iterative model. It provides support for **Risk Handling**. The Spiral Model is a systematic and iterative approach to software development. In its diagrammatic representation, it looks like a spiral with many loops. The exact number of loops of the spiral is unknown and can vary from project to project. Each loop of the spiral is called a phase of the software development process.



### f) Big Bang Model

- Focusing on all resources with no or very little planning
- Used for academic projects
- Used when release date is not given and requirement is unknown

### g) Agile Model

- Agile" refers to the ability to be prompt to change or quick to adapt. In the context of Agile methodology, it emphasizes flexibility, responsiveness, and continuous improvement, allowing teams to adapt to evolving requirements, feedback, and changing conditions efficient.
- In Agile methodology, several frameworks help teams implement Agile principles with structured approaches. For example, Scrum is a widely-used framework, and a combination of Scrum and Kanban, known as Scrumban, is also popular. Nested Scrum is available for complex projects, and Extreme Programming (XP) is another option. When there are dependencies across more than eight Scrum teams, a framework like SAFe (Scaled Agile Framework) is used. Currently, Scrum is the most widely adopted and preferred framework, so we will focus on the Scrum framework.
- Agile Methodology means a practice that promotes continuous iteration of development and testing throughout the SDLC of the project.

- The entire process is broken into small individual models that designers work on.
- Errors can be fixed in the middle of the project.
- Customers has early and frequent opportunity to look at the product.
- Development process is iterative and the project is executed in (2-4) week iterations.
- Every iteration has its own testing.
- Testers and developers work together.
- At the end of every sprint UAT is done.







### ❖ Agile Model Vs Waterfall Model

Agile and Waterfall models are two different methods for the software development process. Though they are different in their approach, both methods are useful at times, depending on the requirement and the type of project.

## 1.3 SOFTWARE TESTING LIFECYCLE (STLC)

The Software Testing Life Cycle (STLC) is a series of phases that guide the testing process to ensure the quality of software.

### 1.3.1 STLC Phases

-  Requirement phase
-  Test planning
-  Test case development
-  Test environment setup
-  Test execution
-  Test cycle closure

- Requirement Analysis:** In this phase the test teams studies the requirement from testing point of view to identify testable requirements.
- Test Planning:** Test planning activities include preparation of test plan or strategy, Test tool selection, Test effort estimation, Resource planning, Training requirement
- Test Case Development:** This phase includes creation, verification and rework of test cases and test scripts after the test plan is ready
- Test Environment setup:** It decides the software and hardware condition under which a work product is tested.

## Test Environments

A test environment is the setup of hardware, software, and configurations used to test an application. It ensures the application works under specific conditions.

- **Development (Dev):** Used by developers for coding and unit testing.
  - **QA:** A stable setup for testers to perform functional, integration, and regression testing.
  - **Pre-production (UAT):** Mimics production for final checks by clients and QA teams before release.
  - **Staging:** A near-identical replica of production for performance and load testing and sanity testing.
  - **Production (Prod):** The live environment where actual users interact with the application.
- e. **Test Execution:** It is carried out by testers based on the test plans and test cases prepared. The process consists of test script execution, test script maintenance and bug reporting.
- f. **Test Cycle Closure:** this phase is completion of test execution and it includes test completion reporting, collection of test completion matrices and test results.

## 1.4 Alphabets of Manual Testing

### 1.4.1 Test Scenario

- A test scenario is a high-level documentation of what needs to be tested.
- It is defined as any functionality that can be tested based on client criteria.
- Test scenarios are written from acceptance criteria of user story.
- It gives an idea of what we need to test.
- E.g.: User Login Functionality.

**Objective :** To verify that a registered user can successfully log into the system with valid credentials and is restricted with invalid credentials.

**Precondition:**

User should be registered and have valid login credentials.

### **1.4.2 Test Case Preparation & Template**

- It is a set of actions/steps executed to verify a particular functionality of the application.
- Test case will have test data, pre-condition, expected and actual results for specific test scenarios.
- It is written from a test scenario.
- Priority of the test case is set based on acceptance criteria of user story.

#### **Contents of Test Case**

1. Release ID
2. User story
3. Test Scenario
4. Test Case Id
5. Test Case Description
6. Test steps
7. Test data
8. Expected result
9. Actual result
10. Status- pass or fail
11. Priority
12. Review/Remarks

### **1.4.3 Test Case Design Techniques**

- It helps to design better test cases. Exhaustive testing is not possible.
- It reduces the no. of test cases to be executed while increasing the test coverage.
- It helps to identify test conditions.

#### **a. Boundary Value Analysis - BVA**

- It is based on testing at the boundaries between the partitions.
- It includes maximum, minimum, inside or outside boundaries, typical values and error values.

- It is based on the concept that if a system works well for a particular value then it will work for all values which come between the two boundary values.

#### **Examples:**

##### ➤ **Age Input Field (1-120)**

A form requires an age between 1 and 120.

1. **Valid boundaries:** 1, 120
2. **Invalid boundaries:** 0, 121

##### ➤ **Password Length (8-16 characters)**

A password field accepts between 8 and 16 characters.

1. **Valid boundaries:** 8, 16
2. **Invalid boundaries:** 7, 17

##### ➤ **Shopping Cart Quantity (1-50 items)**

A shopping cart accepts between 1 and 50 items.

1. **Valid boundaries:** 1, 50
2. **Invalid boundaries:** 0, 51

##### ➤ **Percentage Field (0-100%)**

A percentage field accepts input between 0 and 100.

1. **Valid boundaries:** 0, 100
2. **Invalid boundaries:** -1, 101

#### **b. Equivalence Class Partitioning**

- It divides the set of test condition into partition
- Input domain is divided into classes from which test cases are designed.
- Test cases of a representative value of each class is equal to test of any other value of the same class.

### **Examples:**

#### **➤ Age Input Field (1-120)**

The system requires an age between 1 and 120.

##### **1. Valid partitions:**

- Age between 1 and 120

##### **2. Invalid partitions:**

- Age less than 1 (e.g., -5, 0)
- Age greater than 120 (e.g., 121, 150)

#### **➤ Password Length (8-16 characters)**

A password field accepts between 8 and 16 characters.

##### **1. Valid partitions:**

- Password length between 8 and 16 characters

##### **2. Invalid partitions:**

- Password length less than 8 characters (e.g., 5, 7)
- Password length greater than 16 characters (e.g., 17, 20)

#### **➤ Number of Items in Cart (1-50 items)**

A shopping cart accepts between 1 and 50 items.

##### **1. Valid partitions:**

- Cart quantity between 1 and 50

##### **2. Invalid partitions:**

- Cart quantity less than 1 (e.g., 0, -3)
- Cart quantity greater than 50 (e.g., 51, 60)

➤ **Marks in Exam (0-100)**

An exam score must be between 0 and 100.

**1. Valid partitions:**

- Score between 0 and 100

**2. Invalid partitions:**

- Score less than 0 (e.g., -10, -1)
- Score greater than 100 (e.g., 101, 150)

**c. Decision Table Technique**

This Test case design technique is used when there is multiple input combinations of inputs. Decision table is also known as cause effect table. It is used to test functions which responds to a combination of inputs.

**Examples:**

➤ **Online Shopping Discount Rule**

- **Scenario:** A shopping platform offers discounts based on customer membership and total purchase amount.
- **Conditions:**
  - Is the customer a member? (Yes/No)
  - Is the total purchase > \$100? (Yes/No)
- **Actions:**
  - Apply a 10% discount.
  - No discount.

Rules	Conditions		
	Customer is a member	Total purchase >\$100	Discount applied
Rule 1	Yes	Yes	10%
Rule 2	Yes	No	10%
Rule 3	No	Yes	No
Rule 4	No	No	No

➤ **Login Access Control**

- **Scenario:** A user tries to log in, and access is granted only if the username and password are correct.
- **Conditions:**
  1. Is the username correct? (Yes/No)
  2. Is the password correct? (Yes/No)
- **Actions:**
  1. Grant access.
  2. Deny access.

Rule	Username Correct?	Password Correct?	Access Granted?
1	Yes	Yes	Yes
2	Yes	No	No
3	No	Yes	No
4	No	No	No



➤ **Credit Card Approval**

- **Scenario:** A bank approves or denies a credit card application based on the applicant's age and credit score.
- **Conditions:**
  - Is the applicant over 18? (Yes/No)
  - Is the credit score  $\geq 700$ ? (Yes/No)
- **Actions:**
  1. Approve the application.
  2. Deny the application.

Rule	Applicant Over 18?	Credit Score $\geq 700$	Card approved
1	Yes	Yes	Yes
2	Yes	No	No
3	No	Yes	No
4	No	No	No

➤ **Image Upload Conditions**

- **Conditions:**
  - Is the file size  $\leq 5$  MB? (Yes/No)
  - Is the file format supported (JPG/PNG)? (Yes/No)
  - Is the user logged in? (Yes/No)
- **Actions:**
  - Upload the image.
  - Deny the upload with an error message.

Rule	File Size <= 5MB	FileFormat Supported (JPG/PNG)	User Logged In?	Action
1	Yes	Yes	Yes	Upload image
2	Yes	Yes	No	Deny Upload: Login Required
3	Yes	No	Yes	Deny Upload: Unsupported Format
4	Yes	No	No	Deny Upload: Unsupported Format and Login Req
5	No	Yes	Yes	Deny Upload: File too Large
6	No	Yes	No	Deny upload: File too large andlogin required
7	No	No	Yes	Deny upload: File too large andunsupported format
8	No	No	No	Deny upload: Multiple errors

#### d. Error guessing

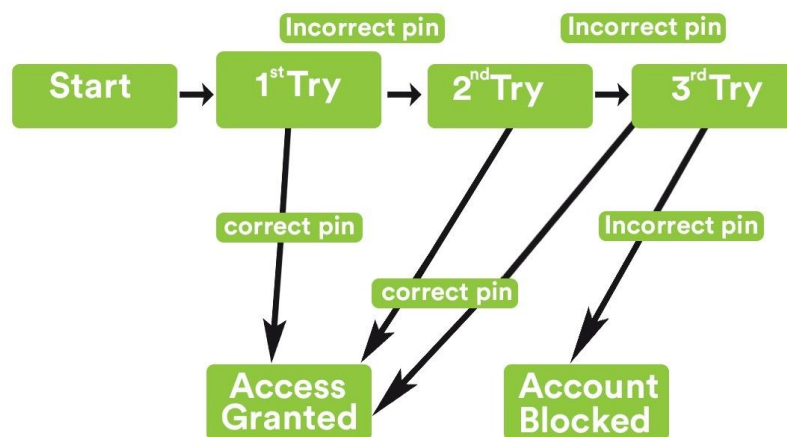
It is done by people having enough experience on the system to guess most likely source of error.

#### e. State Transition Technique

- It is done when changes in input condition change the state of Application Under Test(AUT).
- Tester thus enters various input conditions in a sequence.
- Testing team provides positive as well as negative input test values.
- It is used when testing is done for limited set of input values.
- Thus the testing team tests the sequence of events which happen in the application under test.

**State transition technique** is a software testing method used to verify the behaviour of a system as it moves between different states. This technique helps ensure that the system behaves correctly when transitioning from one state to another, based on certain inputs or events.

### → State Transition Diagram



There are 4 main components of the state transition model as below

1. **State** – e.g.: 1<sup>st</sup> Try – that the software might get
2. **Transition** – From one state to another
3. **Events** – E.g.: Incorrect Pin – That origin a transition like closing a filter or withdrawing a money
4. **Actions** – E. g : Access Granted – Result from a transition



- This main advantage of this testing technique is that it will provide a pictorial or tabular representation of system behaviour which will make the tester to cover and understand the system behavior efficiently.
- States the software can be Represented by rectangles with rounded corners
- A state is a situation or condition of a system. It is represented by a node in a state transition diagram. Each node represents a different state of the system.
- Transitions from one state to another
- Represented by arrows
- A transition is a change in a state of a system that occurs when it responds to an event.
- Events that result in transitions
- Labelled above the applicable transition arrow
- An event is an action or occurrence that triggers a change in the state of a system.
- Actions that result from a transition from an event
- Could be represented by a message box
- An action is a behaviour that the system exhibits when it changes its state.

### State Transition Table

	Correct PIN	Incorrect PIN
S1) Start	S5	S2
S2) 1 <sup>st</sup> attempt	S5	S3
S3) 2 <sup>nd</sup> attempt	S5	S4
S4) 3 <sup>rd</sup> attempt	S5	S6
S5) Access Granted	–	–
S6) Account blocked	–	–

In the table when the user enters the correct PIN, state is transitioned to S5 which is Access granted. And if the user enters a wrong password he is moved to next state. If he does the same 3<sup>rd</sup> time, he will reach the account blocked state.

- **Test Data**

Test data in software testing refers to the input values or conditions used during the testing process to verify whether a software application behaves as expected. This data helps in ensuring the application functions correctly under various scenarios, including both normal and edge cases.

#### **1.4.4 Bug Vs Defect Vs Error Vs Failure**

- A mistake in the code is called an error.
- Error found by the tester is called Bug.
- The bug accepted by the developer is called defect.
- When a product fails in production it is called failure.

#### **1.4.5 Testing Sub Tasks & Activities**

##### **a. Developers Subtask:**

- Dev Clarification
- Code change / Development
- Code review
- Unit testing
- Code deployment - QA Environment

##### **b. Testers Subtask:**

- QA clarification
- Test scenario preparation
- Test scenario review
- Test case preparation
- Test case review
- Test data preparation
- Test case execution

## **1.5 TYPES OF TESTING**

### **1.5.1 Functional Testing:**

Validates that the software functions according to specified requirements. It focuses on user requirements and uses input/output comparisons.

- **Smoke Testing**

Smoke Testing is a software testing process that determines whether the deployed software build is stable or not.

- Testing the basic features of an application
- Smoke testing is the testing done on an initial build
- The basic functionality in the initial build is tested
- Smoke testing is also known as “Build Verification Testing” or “Confidence Testing.”
- Smoke Testing verifies the basic functionality in the initial build

Example - Verify the login is working

- Verify the logout is working
- Verify the package can be added

- **Sanity Testing**

- Sanity testing is the subset of regression testing
- Sanity Testing is done to check the major functionalities

- **Regression Testing**

- Regression is defined as a type of software testing to confirm that a recent program or code change has not adversely affected existing features
- We need to test the Impacted areas
- Regression testing aims to verify that previously working features still work correctly after changes have been made elsewhere in the software.
- Re-executing the already executed test cases

- **Ad-hoc Testing**

- Testing the Application randomly without looking in to the requirements is called Ad-hocTesting
- Ad hoc testing is a software testing technique performed without any specific test plan or predefined set of steps. Instead, testers use their intuition, experience, and creativity to identify defects and issues that more formal testing methods may not find.(called RandomTesting)

- Testing without TC
- Knowledge on both domain and product

- **Exploratory Testing**

- Exploratory Testing is a software testing approach where testers actively explore the application without predefined test cases. They use their experience and creativity to find defects.
- Purpose: To discover unexpected issues and evaluate software quality.
- Method: Testers interact with the application freely, adapting their testing based on what they find.
- Example: A tester navigates through different features of an app, trying various actions to see if any bugs appear.

**Drawbacks of Exploratory Testing:**

- You might misunderstand any feature as a bug (or) any bug as a feature since you do not have requirements.
- Time-consuming.
- If there is any bug in the application, you will never know about it.

### **1.5.2 Non-Functional Testing:**

Assesses aspects like performance, usability, reliability, and security. This includes load testing, stress testing, and scalability testing.

**a. Performance Testing:**

- **Load Testing:** Evaluates the system's behavior under expected load conditions to ensure it can handle the anticipated number of users or transactions.
- **Stress Testing:** Tests the application beyond its normal operational capacity to determine its breaking point and how it handles extreme conditions.

**b. Compatibility Testing:**

- Checks the application's compatibility with various devices, browsers, operating systems, and network environments to ensure consistent performance across platforms.

**c. Security Testing:**

- Assesses the application's security features to identify vulnerabilities and weaknesses. This can include penetration testing, vulnerability scanning, and evaluating the effectiveness of security controls.

**d. Portability Testing**

- Portability testing is a type of non-functional testing that evaluates how easily a software application or system can be transferred from one environment to another. This includes testing the application across different operating systems, hardware platforms, browsers, devices, and network environments.

### **1.5.3 Automation Testing**

- Automation testing is a software testing technique that uses specialized tools and scripts to execute test cases automatically, compare actual results with expected outcomes, and identify defects in the application. It aims to reduce manual effort, improve efficiency, and ensure faster and more accurate test execution.

Automation is particularly useful for repetitive test cases, large-scale testing scenarios, or regression testing where the same tests need to be run repeatedly after changes in the code. Popular automation tools include Selenium, Appium, and JUnit.

**Key Benefits:**

- Saves time and resources.
- Improves test accuracy by minimizing human errors.
- Facilitates testing of complex applications and large datasets.
- Enables continuous integration and continuous testing.



## **Module - 2**

# **Software Testing Complete Guide**

### **2.1 TESTING MYTHS**

#### **2.1.1 Testing is too expensive and time consuming**

Mainly people think that too much money need to be spend for testing but practically, if we test application in a proper way, this will save a lot of money in the maintenance phase of the s/w at a later point of time.

#### **2.1.2 Only fully developed products are tested**

In Testing we do module wise analysis as well as integration of different modules will also be carried out. So, the testing process is not done after the product is completely developed, it is a step-by-step short time interval process.

#### **2.1.3 Complete Testing is Possible**

Myth: Testing can cover every possible scenario and ensure 100% defect detection.

Reality: It is nearly impossible to test every combination of inputs, configurations, and edge cases in complex systems due to time, resource, and budget constraints. Testing focuses on risk-based and priority-driven approaches to maximize coverage within practical limits.

#### **2.1.4 Bug-Free Software**

Myth: Testing guarantees that the final product will be completely free of bugs.

Reality: While rigorous testing reduces the number of defects, it cannot guarantee a completely bug-free product. Some defects may only surface under rare conditions, or after deployment when the system interacts with real-world environments.

### 2.1.5 Missed Defects Indicate Poor Testing

**Myth:** If defects are found after testing, it means the testing team didn't do their job well.  
**Reality:** No testing process can find all defects. Factors such as limited resources, unclear requirements, or unforeseen user behaviors can lead to missed defects. Testing aims to identify the most critical issues that affect functionality and quality, but some low-risk or rare issues might remain undetected.

### 2.1.6 TESTING PRINCIPLES

#### a. Exhaustive testing is not possible

Exhaustive testing is not possible. Instead, we need the optimal amount of testing based on the risk assessment of the application. Unless the application is a very simple one with limited input, it is not possible to test all possible combinations of data and scenarios.

**E.g.:** Testing a form with numerous input fields, combinations of special characters, numbers, and letter inputs for each field would be too time-consuming. Instead, testers use techniques like boundary value analysis to test key scenarios instead of every possible input.

#### b. Defect Clustering

Defect clustering, which states that a small number of modules contain most of the defects detected. It is based on the Pareto principle of software testing: Approx 80% of problems are found in 20% of modules.

**E.g.:** In a banking application, many defects are found in the "payment processing" module because it is the most complex. Therefore, testers focus more efforts on this area, knowing defects tend to cluster in a few key areas.

#### c. Pesticide Paradox

If you keep on running the same test again and again, there won't be any chance of finding new defects. Every time a new function is added, we need to carry out regression testing with the test case must also be modified.

#### d. Testing shows a presence of defects

Testing talks about the presence of defects and doesn't talk about the absence of defects. It reveals one or more defects exist in the application. Testing alone cannot prove the application is error-free.

**E.g.:** Imagine you have an app that displays an error message when the user enters incorrect login credentials. Testing may reveal that the error message doesn't appear for some invalid inputs, indicating a defect.

#### e. Absence of Error – fallacy

It is possible that software which is 99% bug-free is still unusable. This can be the case if the system is tested thoroughly for the wrong requirement. Software testing is not merely finding defects, but also to check that software addresses the business needs.

E.g.: A team tests and fixes all known defects in a time-tracking tool. However, if the tool doesn't meet user needs, it is still considered a failure despite having no bugs. Testing should focus on both finding defects and ensuring the product meets user expectations.

#### **f. Early Testing**

Testing should start as early as possible in the Software Development Life Cycle. So that any defects in the requirements or design phase are captured in early stages. It is much cheaper to fix a Defect in the early stages of testing. This testing is prepared for each level of development cycle.

E.g.: In a project following Agile methodology, testers start testing user stories as soon as they are defined, finding defects during the requirements phase (e.g., missing functionality or unclear requirements) rather than waiting until the development phase, when fixing defects can be more costly.

#### **g. Testing is context dependent**

Testing is context dependent which basically means that the way you test an e-commerce site will be different from the way you test a commercial off the shelf application.

## **2.2 Test Design Techniques**

Already discussed in the previous module

## **2.3 TESTING APPROACHES**

### **2.3.1 Smoke Testing**

- A quick set of tests to check if the **basic functionality** of an application works as expected.
- Performed after a new build is released to ensure critical functionalities are operational.
- **Example:** Checking if the login screen loads and accepts credentials.

### **2.3.2 Sanity Testing**

- Sanity testing is the subset of regression testing
- **Sanity Testing is done to check the major functionalities**
- **Example:** Testing a newly added "Search" feature on a website.

### 2.3.3 Retesting

- Testing specific test cases that previously failed to ensure the issues have been fixed.
- Focused and repetitive for validation.
- **Example:** Retesting a failed "Password Reset" functionality after the bug is resolved.

### 2.3.4 Regression Testing

- Ensures that new changes (like code updates or bug fixes) don't adversely impact existing functionality.
- **Example:** Testing all major workflows after implementing a new "Order Cancellation" feature.

### 2.3.5 Ad-hoc Testing

- Testing the Application randomly without looking in to the requirements is called Ad-hoc Testing
- Ad hoc testing is a software testing technique performed without any specific test plan or predefined set of steps. Instead, testers use their intuition, experience, and creativity to identify defects and issues that more formal testing methods may not find.(called Random Testing)
- Testing without TC
- Knowledge on both domain and product

### 2.3.6 Monkey Testing

- Random and chaotic testing to check how the system behaves under unpredictable inputs.
- Goal: Identify crashes or performance issues.
- **Example:** Entering random characters into a form to see if the system crashes.

### 2.3.7 Exploratory Testing

- Testing without predefined test cases while simultaneously learning about the application.
- Useful for discovering edge cases or areas not covered by traditional testing.
- Testing without TC
- Knowledge only on domain

- **Example:** Exploring a new feature by interacting with it in various ways.

### 2.3.8 Compatibility Testing

- Ensures the application works across different devices, browsers, operating systems, or network environments.
- **Example:** Testing a web app on Chrome, Firefox, and Safari across Windows, macOS, and Linux.

## 2.4 TESTING – QA & QC

### 2.4.1 Verification – Validation

<u>Verification/Static Testing</u>	<u>Validation/Dynamic Testing</u>
It is a <b>process-oriented</b> approach.	It is a <b>product-oriented</b> approach.
It includes checking documents, designs, code etc.	It is a dynamic mechanism of validating the actual product.
It does not involve executing the code.	It always involves executing the code.
It involves methods like review, walk through, inspection and desk checking.	It uses methods like black box testing, white box testing and non-functional testing.
It finds bugs early in development cycle.	It finds bugs that the verification process cannot catch.
Verification is done on SRS doc.	Validation is done on software code.
It is a static testing	It is a dynamic testing.
Eg: Code Review, Testcase Review, Test scenario Review, Design review etc.	Actual Product testing is called validation

## 2.4.2 Quality Assurance and Quality Control (QA & QC)

Quality Assurance	Quality Control
Defect prevention	Defect identification
Avoid defects in the deliverable	Correct defects in the deliverable
A proactive process	A reactive process
Process-based approach	Product-based approach
Quality managing processes	Quality verification
<b>Verification</b> is a QA Activity	<b>Validation</b> is a QC Activity

## 2.4.3 Bug Vs Defect Vs Error Vs Failure

- A mistake in the code is called an **error**.
- Error found by the tester is called **Bug**.
- The bug accepted by the developer is called **defect**.
- When a product fails in production it is called **failure**.

## 2.5 TESTING METHODS

### 2.5.1 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 Behavioral Testing.



## **Examples:**

### ● **Login Functionality Testing:**

- Test whether users can successfully log in with valid credentials and are denied with invalid ones.
- Enter valid username and password to check if the login succeeds. Enter incorrect credentials to verify that the system prevents access.

### ● **Shopping Cart Functionality:**

- Verify that items can be added to or removed from the cart and the total price updates correctly.
- Add 2 products to the cart, check if the total price reflects the sum, and remove 1 product to see if the total adjusts.

### ● **Form Validation Testing:**

- Check if form fields (like name, email, etc.) accept valid inputs and show errors for invalid inputs.
- Enter an invalid email format like "abc.com" to verify if an error message appears, and input "test@example.com" to ensure it's accepted.

### ● **Payment Gateway Testing:**

- Test the payment process to ensure users can successfully complete payments and handle failed transactions.
- Attempt payment with valid credit card details to verify successful transactions and use an expired card to check for failure handling.

### 2.5.2 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, Open box testing, Transparent box testing, Code-based testing, and Glass box testing.

**Examples:**

- **Unit Testing:**

- Test individual functions or methods to ensure they perform as expected.
- Test a function that calculates the sum of two numbers to verify it returns correct results for different inputs.

- **Code Coverage Testing:**

- Ensure that all parts of the code are executed and tested, such as loops, conditions, and branches.
- Test a program with multiple if-else statements to ensure each branch is executed at least once.

- **Path Testing:**

- Test every possible path in the code to ensure that all logic flows are handled correctly.
- Test a function that checks user permissions, making sure that each permission level follows the correct path.

- **Security Vulnerability Testing:**

- Test the code for security loopholes, such as buffer overflows or SQL injection vulnerabilities.
- Check for vulnerabilities in input fields that could allow SQL injection attacks.



### **Types of White Box Testing:**

- 1) **Unit Testing:** Focuses on individual components or functions of the code. Developers usually perform these tests to validate that each unit behaves as expected.
- 2) **Integration Testing:** Tests the interactions between integrated modules to ensure they work together correctly. This can involve testing data flow and communication between modules.
- 3) **Code Coverage Testing:** Measures how much of the code is executed during testing. It identifies parts of the code that are not tested, using metrics like statement coverage, branch coverage, and path coverage.
- 4) **Mutation Testing:** Introduces small changes (mutations) to the code and checks if the existing tests can detect these changes. It helps evaluate the effectiveness of the test suite.
- 5) **Performance Testing:** Analyzes the performance characteristics of the software by examining the code's efficiency, such as response times, resource usage, and scalability.
- 6) **Security Testing:** Involves reviewing the code for vulnerabilities and security flaws. This includes checking for issues like SQL injection, buffer overflow, and improper authentication.
- 7) **Control Flow Testing:** Focuses on the logical flow of the program. Testers evaluate different paths through the code to ensure that all branches and decision points are tested.
- 8) **Data Flow Testing:** Examines the lifecycle of data variables, ensuring that data definitions and usages are properly handled and that there are no undefined or unused variables.

### 2.5.3 Grey Box Testing:

Grey box testing is a software testing technique that combines both black box and white box testing approaches. Testers have partial knowledge of the internal workings of the system, allowing them to test both the functionality and some internal components.



#### Examples:

- **Database Testing:**

- Test how the application interacts with the database by verifying queries, stored procedures, and data integrity.
- Test a search function where you check both the UI response and ensure that the correct SQL query is executed in the backend.

- **Session Management Testing:**

- Check how user sessions are managed, both from the user interface and by inspecting the session data store in the backend.
- Log in as a user, verify the session token is correctly generated, and ensure that session expiration works as expected.

- **Input Validation with Code Review:**

- Validate that the application properly sanitizes user input and handles it securely by reviewing the input validation logic.
- Submit various inputs in a form (like special characters) and verify that the input is both displayed properly in the UI and handled securely in the backend.

- **File Upload Testing:**

- Test the file upload functionality by ensuring files are accepted correctly in the UI and that the backend validates and stores them securely.
- Upload a file through the UI and then verify that the file is stored correctly on the server without errors or security issues.

## 2.6 LEVELS OF TESTING

### 2.6.1 Unit Testing → Done by Developer

Unit Testing is a software testing method where individual units or components of a software are tested in isolation to ensure they function as expected. The goal is to validate that each unit of the software performs correctly without relying on other parts of the system.

**Example:**

- **Login Validation Function**

Checking if a login function correctly validates usernames and passwords. "Success" for valid credentials and "failure" for invalid credentials.

- **Math Function –**

- Test that a function calculating the square root of the number works as expected.
- $(2+3=5)$  - It checks whether the function performs correctly by giving the output as 5.

- **Text Formatter –**

- Ensure that a function formatting a string to title case works as intended. string "hello world" to title case "HELLO WORLD"

- **Discount Calculator -**

- Test that a function applies a discount correctly based on user input.
- Store offering 10% discount for a \$100 product.

### 2.6.2 Integration Testing → Done by Developer and Tester

**Integration Testing** is a type of software testing where individual units or components are combined and tested as a group. The goal is to verify the interactions between different components and ensure that they work together as expected.

**Examples:**

- **Login and Profile Integration -**

After logging in, ensure the correct user profile is displayed based on login credentials.

- **Shopping Cart and Inventory Integration -**

Adding an item to the cart should reduce its quantity in the inventory.

- **Payment Gateway and Order Processing -**

Completing a payment should update the order status to "Paid" and generate an invoice.

- **Email Notification and Registration –**

Completing the registration process should trigger a welcome email to the user.

➤ **Stub and Driver**

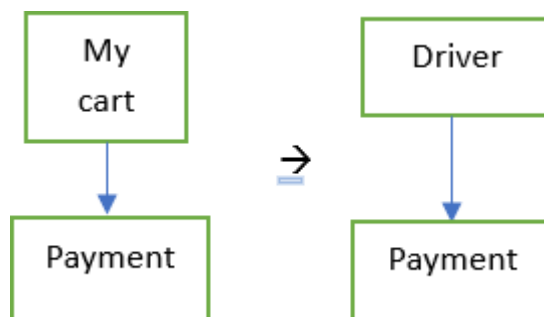
**Stubs and Drivers** are the dummy programs in Integration testing used to facilitate the software testing activity. These programs act as a substitute for the missing models in the testing. They do not implement the entire programming logic of the software module but they simulate data communication with the calling module while testing.

**Stub:** Is called by the Module under Test.

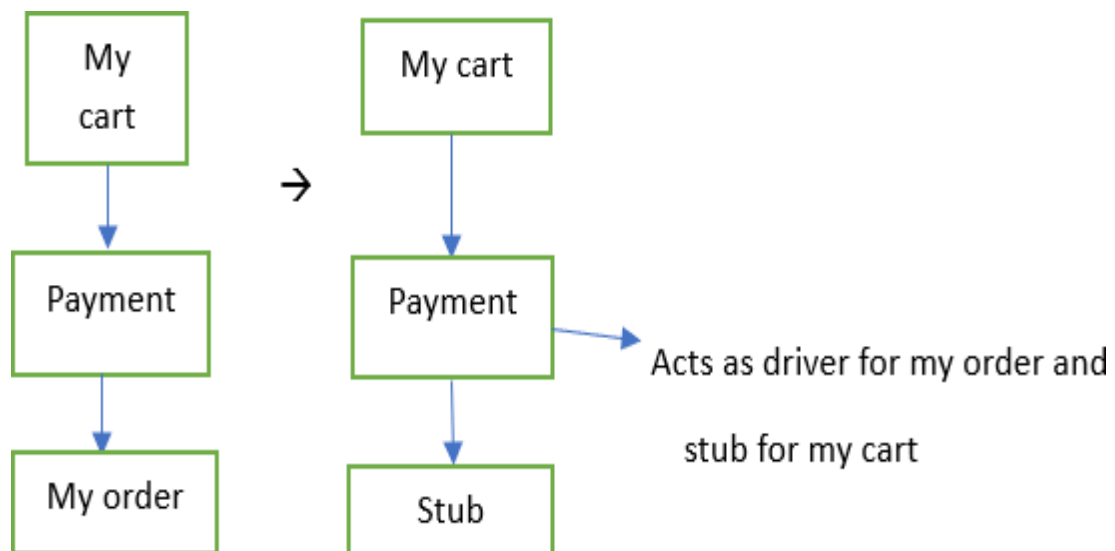
**Driver:** Calls the Module to be tested.

**Eg:**

- When my cart is not ready



- When my order is not ready



- **Bottom-up Integration Testing**

**Bottom-up Integration Testing** is a strategy in which the lower level modules are tested first. These tested modules are then further used to facilitate the testing of higher level modules. The process continues until all modules at top level are tested. Once the lower level modules are tested and integrated, then the next level of modules are formed.

- **Top Down Integration Testing**

**Top Down Integration Testing** is a method in which integration testing takes place from top to bottom following the control flow of software system. The higher level modules are tested first and then lower level modules are tested and integrated in order to check the software functionality. Stubs are used for testing if some modules are not ready.

### 2.6.3 System Testing → Done by Tester

**System testing** is a type of software testing that involves testing a complete and integrated software application as a whole. The purpose of system testing is to validate the end-to-end functioning of the system based on its requirements. It ensures that the system meets the functional and non-functional requirements and behaves as expected in the real-world environment.

### Examples:

- **User Registration Flow -**

- Verify that the entire user registration process works as expected.
- Ensure that entering valid user details, submitting the and receiving confirmationemail works smoothly.

- **Product Purchase Process -**

- Test the entire process of purchasing a product, from adding it to the cart to payment and order confirmation.

- **User Dashboard -**

- Ensure that the user dashboard displays correct and real-time information after login.
- Verify that after logging in, the dashboard correctly displays user profile data, recent activities, and notifications.

- **Multi-language Support -**

- Test the system's ability to switch between different languages based on user preference.
- Ensure that selecting a language (e.g., French or Spanish) updates

#### 2.6.4 Acceptance Testing → Done by clients

**Acceptance Testing** is the final phase of software testing, where the system is tested to ensure it meets the business requirements and is ready for release to the end users. It is conducted to verify whether the software satisfies the acceptance criteria defined by the stakeholders and to ensure it's fit for production use.

There are two types of acceptance testing: Alpha testing and beta testing

##### 1. Alpha Testing

Alpha Testing is a type of acceptance testing; performed to identify all possible issues and bugs before releasing the final product to the end users. Alpha testing is carried out by the testers who are internal employees of the organization.

## **Examples:**

- **Mobile Banking App - Basic Functionality Testing**

A new mobile banking app is tested internally by the development team to check core features such as logging in, viewing account balance, transferring money, and making payments. The purpose is to catch any obvious bugs or **crashes** in a controlled environment before it is released for broader testing.

- **E-commerce Website - Checkout Process**

Before the e-commerce website is released to external testers, the team internally tests the entire checkout process. They simulate adding items to the cart, applying discounts, and making a payment to ensure the critical features function as expected.

- **Educational Software - Course Enrollment Feature**

An educational platform's new course enrollment feature is tested by the internal team. The team checks whether students can successfully enroll in courses, and whether course materials are available for those who have enrolled.

- **Gaming Application - Level Progression**

A gaming app's core feature, such as level progression, is tested by the developers and QA team. They play through several levels of the game to verify the mechanics work as intended, the scoring system functions properly, and no critical bugs are encountered.

## **2. Beta Testing**

Beta Testing is performed by “real users” of the software application in “real environment” and it can be considered as a form of external User Acceptance Testing. It is the final test before shipping a product to the customers. Direct feedback from customers is a major advantage of Beta Testing. This testing helps to test products in customer’s environment.

### **Examples :**

- **Social Media App - User Feedback on New Features**

A new social media app is released to a limited group of real users to test new features like posting stories, messaging, and live streaming. Users report any issues they encounter, such as bugs, performance lags, or unclear interfaces, helping the developers refine the app before a wider launch.

- **Productivity Software - Cross-Platform Compatibility**

A new productivity tool, like a task manager, is released to a group of beta testers to see how it performs across different platforms (Windows, macOS, mobile). Testers use it in their daily work routines, providing feedback on any bugs, interface issues, or missing features across devices.

- **Online Food Delivery App - Real-World Order Testing**

A food delivery app in beta is provided to users who can place real orders from local restaurants. The app’s functionality for browsing menus, making payments, tracking orders, and receiving deliveries is tested in real-world conditions. Users provide feedback on the app’s performance and ease of use.

- **Fitness Tracker - App and Device Integration**

A new fitness tracker device is in beta testing, with users wearing the device and syncing it with a companion app. Beta testers report back on issues like syncing errors, inaccurate data tracking, and battery performance, allowing the company to fine-tune both the device and the app before a full release.



## **2.7 DOCUMENTATION**

### **2.7.1 Test Strategy**

- The Test Strategy document is created by the Project Manager during the initial phase of the project.
- It is a high-level document that defines the testing objectives, scope, approach, and resources from a business perspective.
- The components of a test strategy include objectives and scope, documentation formats, Test processes, team reporting structure, client communication strategy, and so on.
- The test approach serves as a general guide for the project at hand.

#### **Details Included in Test Strategy Document**

The test strategy document includes the following important details:

- Overview and Scope.
- Software and testing work products that can be reused.
- Details about the various test levels, their relationships, and the technique for integrating the various test levels.
- Techniques for testing the environment.
- Level of testing automation.
- Various testing tools.
- Risk Assessment.
- For each level of the test Conditions for both entry and exit.
- Reports on test results.
- Each test's degree of independence.
- During testing, metrics and measurements will be analysed.
- Regression and confirmation testing.
- Taking care of discovered flaws.
- Configuring and managing test tools and infrastructure.
- Members of the Test team's roles and responsibilities.

## Test Strategy Selection

The following factors may influence the test approach selection:

- The test strategy chosen is determined by the nature and size of the organization.
- One can choose a test strategy based on the project needs; for example, safety and security applications necessitate a more rigorous technique.
- The test strategy can be chosen based on the product development model.
- Is this a short-term or long-term strategy?
- Organization type and size.
- Project requirements, Safety and security applications necessitate a well-thought-out strategy.
- Product development model.

### 2.7.2 Test Plan Document

A Test Plan Document is a formal outline that describes the testing objectives, scope, approach, and activities. It acts as a roadmap for the testing team, ensuring alignment with project goals and effective communication among stakeholders. Below are the typical components included in a test plan:

- **Scope:** Defines the boundaries of testing, including what is in-scope and out-of-scope for the testing effort.
- **Test Objectives:** Details the goals of testing, such as validating specific functionalities or ensuring system performance.
- **Test Environment:** Specifies the setup, tools, hardware, and software configurations required to conduct testing.
- **Test Deliverables:** Lists the key artifacts that will be delivered, such as test cases, test scripts, and defect reports.
- **Resources and Responsibilities:** Identifies team members involved in testing and their specific roles.
- **Testing Schedule:** Provides a timeline for the testing activities, including milestones and deadlines.
- **Risk and Mitigation:** Highlights potential risks that could impact testing and suggests strategies to address them.

- **Test Approach:** Describes the methods and techniques to be used, such as manual testing, automated testing, or exploratory testing.
- **Exit Criteria:** Specifies the conditions under which testing will be considered complete, such as achieving a certain test coverage or resolving critical defects.
- **References:** Includes related documents, standards, or guidelines used to prepare the test plan.

### 2.7.3 Test Approach

#### Definition:

The Test Approach outlines the strategy and methodology that the testing team will follow to ensure that the application meets the quality standards. It is a high-level document created during the test planning phase and acts as a guide for all testing activities.

#### Purpose:

- Defines the scope, techniques, and resources required for testing.
- Ensures all team members are aligned with the testing objectives.
- Helps manage risks by addressing challenges early.

#### Key Components:

- Scope of Testing:** Clearly identifies what functionalities, modules, or features will be tested and what will not.
  - Example: Testing the login, registration, and checkout modules of an e-commerce app.
- Testing Techniques:** Specifies the type of testing to be performed.
  - Example: Functional, regression, performance, or security testing.
- Tools and Resources:** Lists the tools, frameworks, and environments needed.
  - Example: Using Selenium for automation and JIRA for defect tracking.
- Roles and Responsibilities:** Defines the roles of team members (e.g., QA Analyst, Automation Engineer).
- Risk Management:** Anticipates potential risks like lack of test data or environment issues and defines mitigation strategies.

## 2.7.4 Test Reports

### Definition:

A Test Report is a document summarizing the results of all testing activities. It provides stakeholders with a snapshot of the testing progress, defects identified, and the overall quality of the application.

### Purpose:

- To provide transparency on testing progress and outcomes.
- To help stakeholders make informed decisions about release readiness.
- To document findings for future reference or audits.

### Structure of a Test Report:

- Introduction:
  - a. Project Name: Name of the application being tested.
  - b. Build Version: Version of the software tested.
  - c. Testing Duration: Start and end dates of testing.
- Test Summary:
  - a. Total Test Cases: Total number of test cases created.
  - b. Passed: Number of test cases that passed.
  - c. Blocked: Test cases that could not be executed due to issues like environment setup.
  - d. Failed: Number of test cases that failed.
- Defect Summary:
  - a. A table showing defect ID, severity, priority, and status (Open, Fixed, Retested).
  - b. Example Table:

Defect ID	Severity	Priority	Status
DEF001	Critical	High	Open
DEF002	Medium	Medium	Retested

- Coverage Metrics:
  - a. Percentage of requirements covered during testing.
  - b. Example: "90% of the requirements were tested during this cycle."
- Observations and Recommendations:
  - a. Highlight any critical risks, unresolved defects, or areas requiring improvement.

### 2.7.5 QA Sign-off

#### **Definition:**

The QA Sign-off is a formal document or email issued by the QA team, confirming that the testing process is complete and the application meets the required quality standards.

#### **Purpose:**

- Acts as an official acknowledgment that the software is ready for production or release.
- Highlights any known risks or limitations for stakeholder awareness.

#### **Key Components:**

- Project Information:
  - a. Project Name: Identify the application.
  - b. Build Version: The version that was tested.
  - c. Environment: Specify if testing was done on staging, production, or development environments.
- d. Testing Summary:
  - e. Total Test Cases: Number executed, passed, failed, or blocked.
- Known Issues: List of unresolved defects with justifications for why they don't block the release.

Example Known Issue: DEF007: Minor UI misalignment on the homepage.

- **Sign-off Statement:** A declaration stating: "The QA team confirms that all critical functionalities have been tested, and the application meets the required quality standards. Known issues, if any, have been communicated to the stakeholders."
- **Approval Section:**
  1. Name, designation, and signature of the QA Lead or Manager.
  2. Date of Sign-off.

Example QA Sign-off Document

**Project Name:**

E-commerce Website

**Build Version:**

v1.2.3

**Environment:**

Staging

**Testing Summary:**

- Total Test Cases: 200
- Passed: 190
- Failed: 5
- Blocked: 5

**Known Issues:**

Defect ID	Description	Priority	Reason for Release
DEF003	Minor alignment issue in the footer	Low	Non-critical, affects only UI.
DEF010	Slow response time during peak hours	Medium	Known limitation to be addressed.

**QA Sign-off Statement:**

The QA team certifies that the application has undergone thorough testing, and all critical functionalities have been validated. Known issues have been communicated to the stakeholders, and the software is deemed ready for release.

Sign-off Approved By:

- Name: Tom Samuel
- Designation: QA Lead
- Date: [Insert Date]

**Features to be tested:**

Highlight the feature under feature Id

## **2.8 AGILE DEVELOPMENT MODEL**

### **2.8.1 Scrum**

Agile is a software development methodology that combines **iterative and incremental** approaches to deliver high-quality software. It includes various frameworks, such as Kanban, Lean, and Extreme Programming (XP) etc. **Scrum**, one of the most popular Agile frameworks, focuses on delivering work in short, time-boxed iterations called **sprints**. Scrum emphasizes roles like the Product Owner, Scrum Master, and Development Team, as well as ceremonies such as Sprint Planning, Daily Standups, Sprint Reviews, and Retrospectives, to ensure continuous improvement and collaboration. Scrum is derived from activity that occurs during a rugby match. Scrum concentrates on working in small teams (9-11) members.

It consists of three roles, and their responsibilities are explained as follows:

- **Scrum Master**

Scrum Master is responsible for setting up the team, sprint meeting and removes obstacles to progress

- **Product owner**

The Product Owner creates product backlog, prioritizes the backlog and is responsible for the delivery of the functionality at each iteration

- **Development Team**

The Development Team in Scrum typically includes a cross-functional group of professionals such as testers, developers, business analysts (BAs), and architects. They work collaboratively to deliver a potentially shippable product increment at the end of each sprint.

### **2.8.2 Product Backlog**

This is a repository where requirements are tracked with details on the no. of requirements (user stories) to be completed for each release.

It should be maintained and prioritized by the Product Owner, and it should be distributed to the scrum team. PO keeps the user story in the product backlog.



### 2.8.3 Scrum Practices

Practices are described in detailed:

#### Process flow of Scrum Methodologies:

Process flow scrum testing is as follows:

- Each iteration of a scrum is known as **Sprint**
- Product backlog is a list where all details are entered to get the end-product
- During each Sprint, top user stories of Product backlog are selected and turned into Sprint backlog
- Team works on the defined sprint backlog
- Team checks for the daily work
- At the end of the sprint, team delivers product functionality

#### Sprint

Sprint is the predefined interval of the time frame in which the work has to be completed.

1 sprint = 4 weeks: The team completes a chunk of work in 4 weeks.

1 release = 1 sprint: At the end of each sprint, the team delivers a fully functional product or update to users, meaning every sprint results in a release.

### 2.8.4 Scrum Meetings

- a. Sprint grooming
- b. Sprint planning meeting
- c. Daily stand-up meeting
- d. Sprint review meeting
- e. Sprint retrospective meeting

#### **a. Sprint Grooming**

It is a pre-planning meeting. Here the team tries to write down the dependencies or other factors which they would like to discuss in a planning meeting.

#### **b. Sprint planning meeting**

- It is the starting point of sprint
- Scrum team gathers
- Scrum master selects a user story based on priority from the product backlog
- Team identifies the task along with the effort (in hrs) needed to complete the user story implementation.

#### **c. Daily stand-up meeting**

Every day the scrum team meets for not more than 15 minutes.

**States three point-**

- What did the team members do yesterday?
- What did the team member plan to do today?
- Any impediments (road blocks)?
  - Scrum master facilitates this meeting
  - In case any team member is facing any kind of difficulty the scrum
  - master follows up to get it resolved.

#### **d. Sprint Review meeting/ Demo**

- At the end of every sprint cycle
- Scrum team meets again and demonstrates implemented user stories to the PO.
- PO cross verifies the story as per its acceptance criteria.
- Scrum master presides over the meeting
- Sprint is closed and tasks are marked done

#### **e. Sprint Retrospective meeting**

- It is done after the review meeting
- Scrum team meets, discuss and document the following points-
- What went well during the sprint (best)?
- What did not go well in the sprint?
- Lessons learned
- Action items
- It thus helps in the continuous improvement of the scrum process.

#### **Scrum Poker/ Planning Pocker**

Scrum Poker, also known as Planning Poker, is a collaborative **estimation technique** commonly used in agile teams to estimate the effort required for completing tasks, typically measured in story points.

##### **How Scrum Poker Works:**

##### **1. Preparation:**

- The team reviews the backlog items to be estimated.
- Each team member is given a set of "cards" (these can be physical cards, online tools, or a mobile app), usually with Fibonacci numbers (1, 2, 3, 5, 8, 13, etc.) or similar sequences that represent effort sizes.

##### **2. Story Presentation:**

- The Product Owner or relevant stakeholder describes a user story or backlog item.
- The team discusses any questions or clarifications needed to understand the scope of work.

##### **Estimation:**

- Each team member privately selects a card that represents their estimate of the story's effort.
- The team reveals their estimates simultaneously.

##### **Discussion:**

- If all estimates are the same, that estimate is usually accepted.
- If there is a wide range, team members discuss their reasoning, often starting with those who picked the highest and lowest estimates. This allows for different perspectives and clarifies any misunderstandings.

- ❖ **Re-estimation** (if needed):
- ❖ After discussion, the team repeats the estimation process until a consensus is reached on a single estimate.

## **Sprint Backlog**

Based on priority user stories are taken from the product backlog. It includes a list of all user stories which the scrum team works on a particular sprint.

## **EPIC**

They are features which are required to be implemented in future whose details are not yet known.

Epic: 3-6 months. E.g.: My cart

|

Feature: E.g.: Adding products to my cart

|

User story: E.g.: Adding product from different locations like wish list, offer zone etc.

## **User Story**

How a user is going to define the requirement. They are the requirements or features which have to be implemented. In scrum, requirements are defined in a single para as we don't have huge requirement documents.

The following format is used

**As a user**

**I want to <some achievable goal / target>**

**So that I <some results or reason for doing the thing>**

**E.g.: As a user, I want to** select a product **So that** I can buy it later.

It is the responsibility of the scrum master and BA to make sure that the PO has drafted the user stories correctly with a proper set of acceptance criteria.

Acceptance criteria: Its further details down the user story.

## **Contents of user story-**

**a. Summary** – E g: [RESTAURANT][ ADD PRODUCT]

**a. DOD- Definition of done.**

Syntax - As a user,

I want to <some achievable goal / target>

So that I <some results or reason for doing the thing>

**E.g.:** As a user I want to add a product to the list so that I can view the product in the products list.

**b. Acceptance Criteria**

E.g.: I want to sort by suppliers list. It should be given on the top right

I want to sort by product type. It should be given on the top right

I want to sort by number of entries. It should be given on the top left.

**To facilitate practical learning, we are using a demo project. Please co-ordinate with the respective trainer regarding the project.**

### **2.8.5 Story Points**

They are the quantitative indication of the complexity of a user story. Based on story point efforts are determined. User story is assigned a story point based on the Fibonacci series (1,2,3,5,8,13,21). Higher the number, the more complex the story. Complexity consists of both development and testing. The Development and testing team must collectively agree to one story point.

**Story point calculation:**

- Team - 5
- Working time - 8 hrs.
- Daily effort - 40 hrs. (5 members \* 8 hours)
- Sprint days - 20 days (4 weeks) - 5 days working
- 1 sprint total effort - 800 hrs. (20 days \* 40 hrs.)
- 1 story point - 8 hrs.
- 1 sprint total story points - 100 points - 20 points meetings
- Total points can be considered - 80 points
- In these 100 points, 20 points for meetings
- Total story points = 100-20=80 points

## Sub Task of a User Story

### Developers Subtask:

- Dev Clarification
- Code change / Development
- Code review
- Unit testing
- Code deployment - QA Environment

### Testers Subtask:

- QA clarification
- Test scenario preparation
- Test scenario review
- Test case preparation
- Test case review
- Test data preparation
- Test case execution

## Spill Over

It is the process of carrying over the task not completed in the previous sprint.

## Burndown chart

- It is used to find the work left to do
- Burn down chart is a graphical representation of effort/work left to do Vs time
- Actual effort may increase or decrease
- Estimated effort should decrease and reach 0.

## 2.9 Test Management:

It specifies the tools used for testing

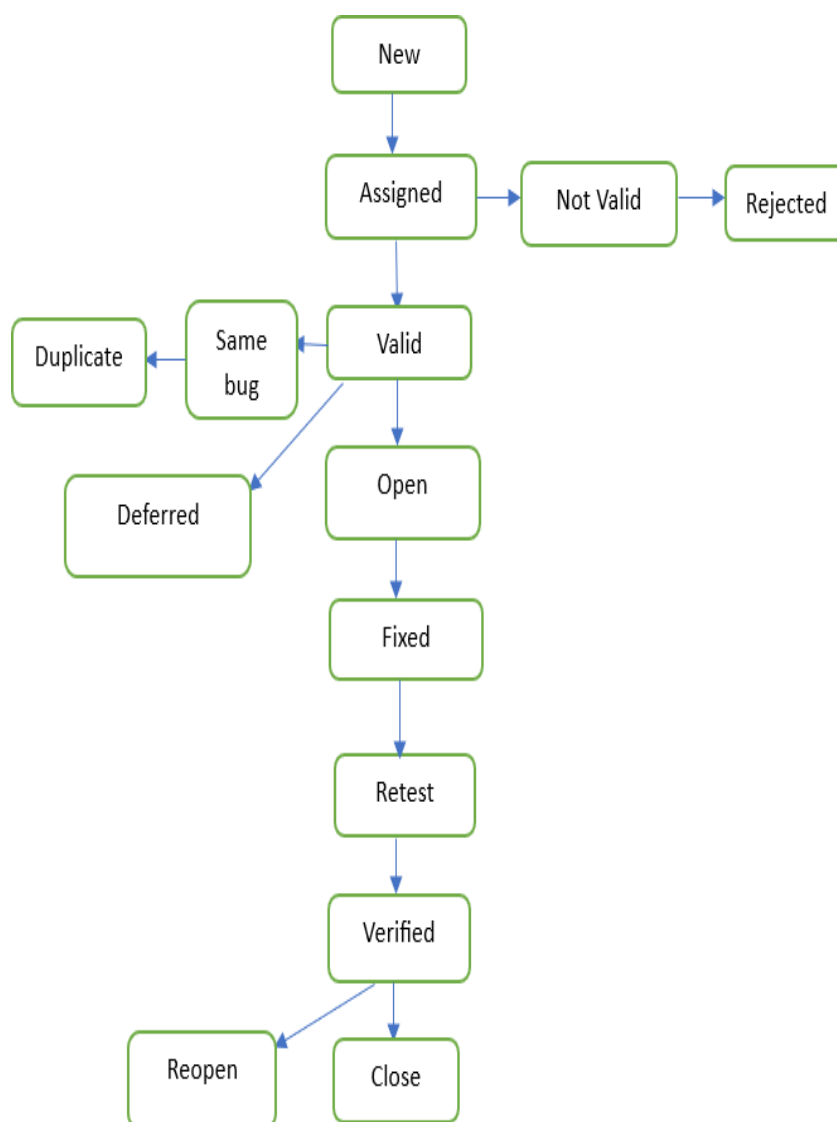
E.g.: Jira

### 2.9.1 Bug Life Cycle

It gives the specific set of states that a bug goes through from discovery to fixation.

- **New:** When a new defect is posted for the first time, it is assigned a status as new.
- **Assigned:** Bug approved by lead tester and assigned to developer team.
- **Open:** Developer starts analyzing and works on the defect.
- **Fixed:** Developer makes a code change and can make bug status as fixed.
- **Retest:** Tester does the retest of the code to check if the defect is fixed.

- **Verified:** Tester retests the bug after it gets fixed and changes the status to verified if no bugs detected.
- **Reopen:** If the bug is present even after the developer has fixed the bug.
- **Close:** If a bug no longer exists then the tester assigns the status as closed.
- **Duplicate:** If the defect is repeated twice or corresponds to the same concept of bug.
- **Rejected:** If the developer feels the defect is not genuine.
- **Deferred:** If the present bug is not of prime priority and is expected to get fixed in the next release.
- It is given / assigned priority by the product owner.
- The Sprint backlog also holds deferred bugs.



### 2.9.2 Bug Reporting & Tracking

While reporting the bug to the developer it must have the following:

- Defect Id
- Defect description/summary
- Version
- Steps to reproduce
- Attachments
- Severity - Set by tester
  - Blocker - Severity 1
  - Critical - Severity 2
  - High - Severity 3
  - Medium - Severity 4
  - Low - Severity 5
- Bug which affects the functionality of an application is given the highest priority.
- Priority - Set by product owner-
  - Urgent - p1
  - High - p2
  - Medium - p3
  - Low - p4
- Work around - Y/N (Is there any alternative solution?)
- Date raised
- Detected by
- Status
- Fixed by
- Date closed
- Release Id
- Sprint
- User story Id



### 2.9.3 Jira – Project Management Tool

#### Management Tool for Software Testers in Agile Projects

##### **Overview:**

Jira is a powerful project management and issue-tracking tool used by Agile teams to manage software development and testing processes. It helps testers efficiently handle their testing tasks, track bugs, and ensure quality within Agile workflows.

Key Features for Testers in Agile Projects:

##### **Bug Tracking:**

- **Log Bugs:** Report and track bugs with detailed information and severity levels.
- **Workflow Management:** Use custom workflows specific to bug resolution, such as Open, In Progress, In Testing, and Closed.

##### **Test Case Management:**

- **Create Test Cases:** Document test cases, link them to user stories, and manage their execution.
- **Integration:** Use plugins like Zephyr or Xray for enhanced test case management within Jira.

##### **Agile Boards:**

- **Scrum Boards:** Track testing tasks and bugs in sprints, moving items through stages like To Do, In Progress, and Done.

##### **Test Execution:**

- **Sprint Testing:** Plan and execute test cases within sprints, ensuring all user stories are thoroughly tested.

##### **Reporting and Metrics:**

- **Bug Reports:** Generate reports on bug trends, test execution results, and coverage.
- **Dashboards:** Create dashboards to monitor testing progress and key metrics in real-time.

## 2.9.4 Severity & Priority

### Priority :

- Priority is defined as How soon the defect has to be fixed or How it impact the business
- Defect priority is set by the product owner.
- Priority is categorized into : low, medium and high
- Priority is driven by business value while Severity is driven by functionality.
- Low: The Defect is an irritant but repair can be done once the more serious Defect has been fixed
- Medium: During the normal course of the development activities, defects should be resolved. It can wait until a new version is created
- High: The defect must be resolved as soon as possible as it affects the system severely and cannot be used until it is fixed
- Priority is for the business

### Severity :

- Severity is defined as ' How much defect impact the system'
- QA engineer determines the severity level of the defect
- Severity is driven by functionality
- Its classified in to Critical,High, Medium and Low
- Critical : Application is completely down
- High : A feature is completely down
- Medium : Feature is partially down
- Low : Small UI Issue
- Severity is for the application

## 2.9.5 Test coverage Management (RTM)

### Traceability Matrix

Traceability matrix is a table type document that is used in the development of software application to trace requirements. It can be used for both forward (from Requirements to Design or Coding) and backward (from Coding to Requirements) tracing. It is also known as Requirement Traceability Matrix (RTM) or Cross Reference Matrix (CRM).

It is prepared before the test execution process to make sure that every requirement is covered in the form of a Test case so that we don't miss out any testing. In the RTM document, we map all the requirements and corresponding test cases to ensure that we have written all the test cases for each condition.

Consider a real-time example for an online banking application, focusing on the “Account Transfer” feature.

**Scenario:** You’re managing the testing of a new “Account Transfer” feature. The feature allows users to transfer money between their accounts and requires a set of functional requirements to be implemented and tested.

### Requirements:

- **REQ-001:** Users should be able to transfer funds from one account to another.
- **REQ-002:** The system should validate that the source account has sufficient funds before processing the transfer.
- **REQ-003:** The system should generate a confirmation message after a successful transfer.

### Test Cases:

- **TC-001:** Verify that users can transfer funds between two accounts.
- **TC-002:** Verify that the system checks for sufficient funds before allowing the transfer.
- **TC-003:** Verify that a confirmation message is displayed after a successful transfer.

### Traceability Matrix Example:

Requirement ID	Requirement Description	Test Case ID	Test Case Description	Status
REQ-001	Users should be able to transfer funds between accounts	TC-001	Verify the transfer functionality between accounts	Passed
REQ-002	The system should validate sufficient funds before transfer	TC-002	Verify fund validation for transfer	Failed
REQ-003	The system should generate a confirmation message after transfer	TC-003	Verify confirmation message after transfer	Passed

- **REQ-001:** This requirement ensures that the transfer functionality works as intended. Test case TC-001, which verifies this functionality, has passed, indicating that the feature works correctly.
- **REQ-002:** This requirement checks if the system validates sufficient funds before allowing a transfer. Test case TC-002 has failed, suggesting a potential issue where the system might not be correctly validating account balances.
- **REQ-003:** This requirement ensures that a confirmation message is displayed after a successful transfer. Test case TC-003 has passed, confirming that the system generates the correct message.

### Why Testers Use a Traceability Matrix

1. **Ensure All Requirements Are Tested:** The matrix helps ensure that every requirement is covered by one or more test cases, preventing incomplete testing.
2. **Identify Gaps and Overlaps:** By mapping requirements to test cases, testers can identify if any requirements are not being tested or if there are redundant test cases.
3. **Manage Changes Effectively:** When requirements change, the matrix helps determine which test cases need to be updated or re-executed, ensuring that changes are properly validated.
4. **Facilitate Communication and Reporting:** The matrix provides a clear, organized view of test coverage, which can be shared with stakeholders to demonstrate that requirements are being met.
5. **Improve Testing Efficiency:** It helps in prioritizing test cases based on requirements and managing the testing effort more effectively.

## Types of Traceability Test Matrix

The traceability matrix can be classified into three different types which are as follows:

- Forward traceability
- Backward or reverse traceability
- Bi-directional traceability

### Forward traceability

The forward traceability test matrix is used to ensure that every business's needs or requirements are executed correctly in the application and also tested rigorously. The main objective of this is to verify whether the product developments are going in the right direction. In this, the requirements are mapped into the forward direction to the test cases.

### Backward or reverse traceability

The reverse or backward traceability is used to check that we are not increasing the space of the product by enhancing the design elements, code, test other things which are not mentioned in the business needs. And the main objective of this that the existing project remains in the correct direction. In this, the requirements are mapped into the backward direction to the test cases.

### Bi-directional traceability

It is a combination of forwarding and backward traceability matrix, which is used to make sure that all the business needs are executed in the test cases. It also evaluates the modification in the requirement which is occurring due to the bugs in the application.

## 2.9.6 Metrics and Measures

In software testing, **metrics** and **measures** play a crucial role in evaluating the quality, efficiency, and progress of testing activities. They provide quantifiable insights into the testing process, helping teams make data-driven decisions and improve overall testing effectiveness.

### a. What Are Metrics and Measures?

- **Measures:** These are raw data points or individual quantitative values.
  - Example: Number of test cases written, number of defects found.
- **Metrics:** These are derived from measures and provide meaningful information for analysis and decision-making.
  - Example: Test Case Pass Rate =  $(\text{Number of test cases passed} / \text{Total test cases executed}) \times 100$ .

## b. Types of Metrics in Software Testing

### Test Execution Metrics

These metrics track the progress and results of the test execution phase.

- **Test Case Execution Rate:**  
**Formula:**  
 $\text{Execution Rate} = \text{Number of Test Cases Executed} / \text{Total Test Cases}$
- **Example:** 70 test cases executed out of 100 → 70%.

### Quality Metrics

These assess the overall quality of the product and testing process.

- **Requirements Coverage:**  
**Formula:**  
 $\text{Requirements Coverage} = \text{Number of Requirements Tested} / \text{Total no of requirements} * 100$
- Example:** 90 requirements tested out of 100 → 90%.

### Defect Metrics

- Defect rejection ratio (DRR) = (No. of defect rejected / total no. of defect raised) \* 100.
- Defect leakage ratio (DLR) = (No. of defect missed / total no. of defects) \* 100.
- The smaller the value of DRR & DLR, the better the quality of test execution. Note:
- If an already corrected bug reappears in the product when the product is updated, the code is new, and hence new bug must be introduced.
- If an already corrected bug reappears before product release/production we reopen the bug, code is the same.

### **Benefits of Metrics in Testing**

- **Improved Decision-Making:** Helps teams prioritize tasks based on data.
- **Progress Tracking:** Provides insights into whether testing is on schedule.
- **Risk Management:** Identifies areas with high defect density or low coverage.
- **Continuous Improvement:** Highlights inefficiencies and areas for optimization.

### **Challenges in Using Metrics**

- **Data Accuracy:** Inconsistent or inaccurate data can mislead teams.
- **Over-reliance on Numbers:** Metrics may not always reflect quality; context is essential.
- **Time-Consuming:** Collecting and analyzing metrics can take significant effort.

### 2.9.7 QA Team Structure – Different Roles and Responsibilities

- **QA Manager/Director:**
  - Responsible for overseeing the entire QA process.
  - Sets QA goals, strategies, and priorities aligned with the organization's objectives.
  - Manages resources, budgets, and timelines for QA activities.
  - Acts as a liaison between QA team and other departments, such as development and product management.
  
- **QA Leads/Team Leads:**
  - Lead smaller groups or teams within the QA department.
  - Coordinate daily activities, assign tasks, and monitor progress.
  - Provide guidance, support, and mentorship to QA engineers.
  - Report to the QA manager/director on team performance and project status.
  
- **QA Engineers/Testers:**
  - Responsible for executing testing activities to ensure software quality.
  - Develop test plans, test cases, and test scripts based on project requirements.
  - Conduct various types of testing, including functional, regression, integration, and performance testing.
  - Report defects, track their resolution, and verify fixes.
  - Collaborate with developers, product managers, and other stakeholders to identify and address quality issues.
  
- **Automation Engineers:**
  - Develop and maintain automated test scripts and frameworks.
  - Identify opportunities for test automation and implement automated testing solutions to improve efficiency and coverage.
  - Collaborate with QA engineers to integrate automated tests into the continuous integration/continuous deployment (CI/CD) pipeline.
  - Monitor and analyse automated test results, identify trends, and recommend improvements.

## 2.10 Other Testing Approaches

### a. Smoke Testing

- A quick set of tests to check if the **basic functionality** of an application works as expected.
- Performed after a new build is released to ensure critical functionalities are operational.
  
- Example: Checking if the login screen loads and accepts credentials.

**b. Sanity Testing**

- Sanity testing is the subset of regression testing
- Sanity Testing is **done to check the major functionalities**
- Example: Testing a newly added "Search" feature on a website.

**c. Retesting**

- Testing specific test cases that previously failed to ensure the issues have been fixed.
- Focused and repetitive for validation.
- Example: Retesting a failed "Password Reset" functionality after the bug is resolved.

**d. Regression Testing**

- Ensures that new changes (like code updates or bug fixes) don't adversely impact existing functionality.
- Example: Testing all major workflows after implementing a new "Order Cancellation" feature.

**e. Ad-hoc Testing**

- Testing the Application randomly without looking in to the requirements is called Ad-hoc Testing
- Ad hoc testing is a software testing technique performed without any specific test plan or predefined set of steps. Instead, testers use their intuition, experience, and creativity to identify defects and issues that more formal testing methods may not find.(called Random Testing)
- Testing without TC
- Knowledge on both domain and product

**f. Monkey Testing**

- Random and chaotic testing to check how the system behaves under unpredictable inputs.
- Goal: Identify crashes or performance issues.
- Example: Entering random characters into a form to see if the system crashes.

**g. Exploratory Testing**

- Testing without predefined test cases while simultaneously learning about the application.
- Useful for discovering edge cases or areas not covered by traditional testing.
- Testing without TC
- Knowledge only on domain
- Example: Exploring a new feature by interacting with it in various ways.



#### **h. Compatibility Testing**

- Ensures the application works across different devices, browsers, operating systems, or network environments.
- Example: Testing a web app on Chrome, Firefox, and Safari across Windows, macOS, and Linux.

## Module – 3

### Test Metrics/Measures and Estimation Detail Study

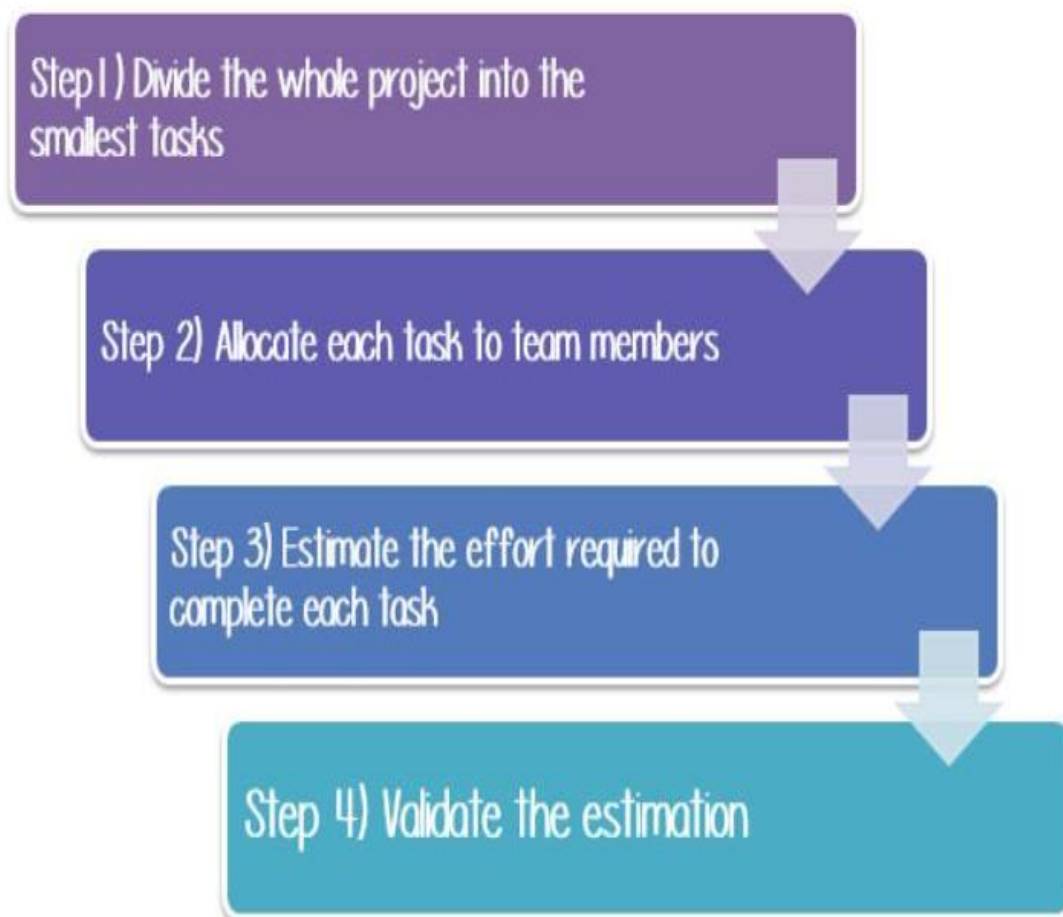
#### 3.1 TEST ESTIMATION

Test estimation is the process of predicting the effort, time, and resources required to complete the testing phase of a project. It helps in planning, scheduling, and allocating the necessary resources to ensure a smooth testing process.

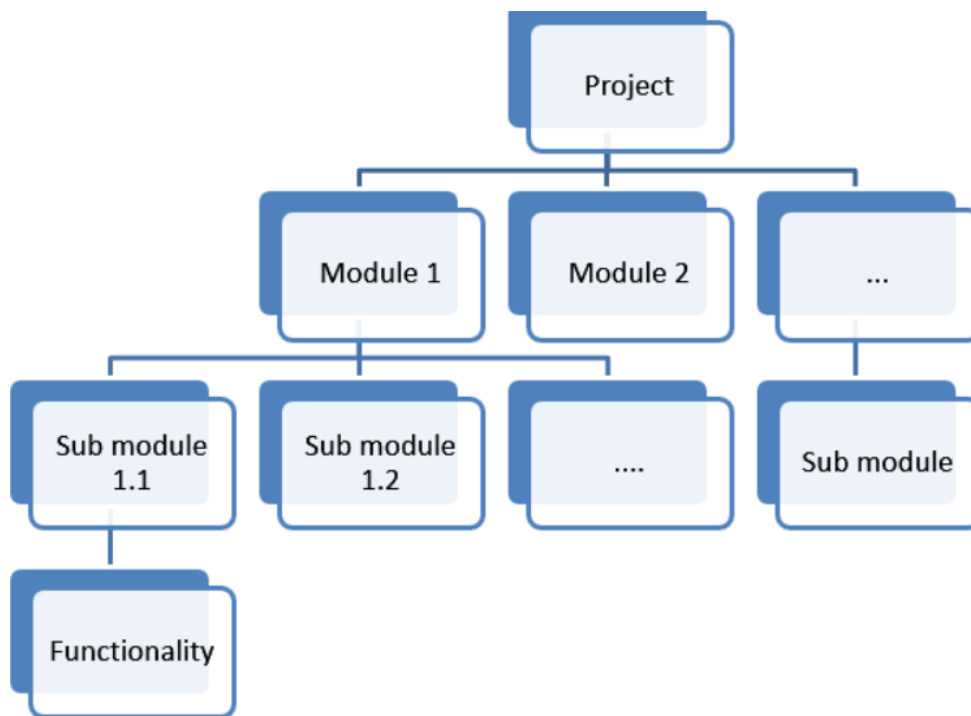
##### 3.1.1 Work Breakdown Structure

Breaking down the testing work into smaller tasks and estimating each.

In this technique, a complex project is divided into **modules**. The modules are divided into **sub- modules**. Each sub-module is further divided into **functionality**. It means divide the whole project task into the **smallest** tasks.



### Step 1) Divide the whole project task into subtasks



After that, you can break out each task to the subtask. The purpose of this activity is create task as detailed as possible

Task	Sub task
Analyze software requirement specification	Investigate the soft requirement specs
	Interview with the developer & other stakeholders to know more about the website
Create the Test Specification	Design test scenarios
	Create test cases
	Review and revise test cases
Execute the test cases	Build up the test environment
	Execute the test cases
	Review test execution results
Report the defects	Create the <b>Defect</b> reports
	Report the defects

## Step 2) Allocate each task to team member

In this step, each task is assigned to the appropriate member in the project team. You can assigned task as follows.

Task	Members
Analyze software requirement specification	All the members
Create the test specification	Tester/Test Analyst
Build up the test environment	Test Administrator
Execute the test cases	Tester, Test Administrator
Report defects	Tester

## Step 3) Effort Estimation For Tasks

There are 2 Techniques which you can apply to estimate the effort for tasks

- Functional Point Method
- Three Point Estimation

## Step 4) Validate the estimation

Once you create an aggregate estimate for all the tasks mentioned in the WBS, you need to forward it to the management board, who will review and approve it.

The management board will review and discuss your estimation plan with you. You may explain your estimation **logically** and **reasonably** so that they can approve your estimation plan.

### 3.1.2 Three Point Estimation

- Three-Point estimation is one of the techniques that could be used to estimate a task. The simplicity of the Three-point estimation makes it a very useful tool for a Project Manager that who wants to estimate.
- In three-point estimation, **three** values are produced initially for every task based on **prior experience** or **best-guesses** as follows



- The **best case** to complete this task is **120 man-hours** (around 15 days). In this case, you have a talented team, they can finish the task in smallest time.
- The **most likely case** to complete this task is **170 man-hours** (around 21 days). This is a normal case, you have enough resource and ability to complete the task
- The **worst case** to complete this task is **200 man-hours** (around 25 days). You need to perform much more work because your team members are not experienced.

### 3.1.3 Delphi Technique

The Delphi Technique is a collaborative estimation method where a group of experts provide estimates anonymously to avoid bias.

Here's how it works:

1. **Initial Estimate:** Each expert independently provides an estimate for the task without discussing it with others.

2. **Review and Revise:** The estimates are shared anonymously, and experts can review each other's inputs.
3. **Rounds of Estimation:** The process is repeated for several rounds, with experts adjusting their estimates after each round based on group feedback.
4. **Consensus:** The rounds continue until the estimates converge to a consensus or a sufficiently close range.

### 3.1.4 Test Case Point Estimation

Test Case Point Estimation is a method for estimating the effort and time needed for testing based on the number and complexity of test cases. Instead of estimating testing time by task (like writing, executing, or reviewing test cases), this technique assigns a point value to each test case depending on its complexity and required resources.

#### Steps in Test case point Estimation

##### 1. Classify Test Cases by Complexity:

Group test cases into categories (simple, medium, and complex) based on factors like the steps involved, data dependencies, and interactions with other modules.

##### Assign Points:

- **Simple** test cases: Usually assigned 1 point, covering basic functionalities with straightforward steps.
- **Medium** test cases: Assigned 2 points, involving moderate complexity or dependencies.
- **Complex** test cases: Assigned 3 points, dealing with advanced functionality, multiple data variations, or integrations.

##### Calculate Effort:

- Add up the points from all test cases to get the total "test case points."
- Multiply by a predefined productivity rate (e.g., time per point) to determine the total estimated effort.

## **3.2 TEST ENVIRONMENT SETUP**

### **3.2.1 Entry Criteria**

**Entry Criteria's** are the conditions or requirements that must be met before testing can begin. They define when we are expected to *start testing*.

#### **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.

### **3.2.2 Exit Criteria**

Exit Criteria defines the items that must be completed before testing can be concluded or when are expected to *stop testing*.

#### **Exit Criteria**

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

#### **Suspension Criteria**

Conditions that determine when testing activities should be halted.

### **3.2.3 Test Bed**

A Test Bed in Software Testing is a software development environment. It allows developers to test their modules without affecting the live production servers. Test bed is not only confined to developers but also used by testers.

### **3.2.4 Test Suite**

A test suite is a collection of test cases grouped together to validate the functionality, performance, or other aspects of a specific feature, module, or the entire system. Test suites are typically organized based on related functionality, testing objectives, or system components and can include both manual and automated test cases.

**Example:**

- **Functional Test Suite:** Includes test cases to verify the main functions of the application.
- **Regression Test Suite:** Includes test cases to ensure that new changes haven't introduced defects into previously working features.
- **Performance Test Suite:** Focuses on test cases that measure the application's speed, stability, and scalability.

### **3.3 ROOT CAUSE ANALYSIS**

Root Cause Analysis (RCA) is a method used to identify the underlying reason for a problem or defect. Instead of just addressing the symptoms, RCA focuses on finding the fundamental cause to prevent recurrence.

#### **3.3.1 Why root cause analysis?**

Root Cause Analysis (RCA) is essential because it helps organizations prevent recurring issues, improve processes, and enhance overall quality. By identifying and addressing the root cause of a problem, RCA helps:

1. **Reduce costs:** Preventing recurring issues saves time and money on repeated fixes.
2. **Improve quality:** Resolving underlying problems improves product and service quality.
3. **Increase customer satisfaction:** Consistently addressing root causes leads to better outcomes, positively impacting customer experiences.
4. **Enhance efficiency:** RCA optimizes processes, reducing delays and increasing operational efficiency.
5. **Build a proactive culture:** Encourages a mindset of continuous improvement by solving problems at their source rather than treating symptoms.

#### **3.3.2 Fish Bone analysis**

Fishbone Analysis, also known as the **Ishikawa Diagram** or **Cause-and-Effect Diagram**, is a tool used in Root Cause Analysis to visually map out the potential causes of a problem.

It helps teams identify, explore, and categorize the possible factors contributing to a specific issue.

#### **Structure of the Fishbone Diagram -**

The diagram resembles a fish skeleton:

1. **Head:** Represents the problem statement or effect.
2. **Bones:** Major categories of potential causes, often classified as:



- **Man (People):** Human-related causes, like skills, experience, or communication.
- **Machine:** Equipment or technology issues.
- **Method:** Process-related factors.
- **Material:** Quality or availability of materials.
- **Measurement:** Data, metrics, or testing issues.
- **Environment:** External factors like physical or organizational conditions.

### Steps to Conduct Fishbone Analysis

- **Identify the Problem:** Define the issue clearly and place it at the "head" of the fish.
- **Determine Major Categories:** Select categories relevant to the problem.
- **Brainstorm Causes:** Within each category, brainstorm specific causes.
- **Analyze the Diagram:** Review and analyze each cause to identify the root cause.
- **Implement Solutions:** Develop and apply solutions to address the root cause(s) identified.

### 3.3.3 Five whys technique

The **5 Whys technique** is a simple but effective Root Cause Analysis tool that involves asking "Why?" repeatedly—usually five times—to drill down to the underlying cause of a problem. This approach helps uncover the root issue rather than just treating symptoms.

#### Steps to Perform the 5 Whys

- **Identify the Problem :** Clearly state the issue you're facing.
- **Ask the First "Why?" :** Determine why the problem occurred.
- **Ask Subsequent "Whys" :** Continue asking "Why?" based on each previous answer.
- **Stop When the Root Cause is Found :** Typically, this is reached within five "Whys," though sometimes fewer or more may be needed.
- **Take Corrective Action:** Address the identified root cause to prevent recurrence.

#### Example

**Problem:** A product shipment was delayed.

1. **Why** was the shipment delayed?
  - Because the product wasn't ready in time.
2. **Why** wasn't the product ready?
  - Because the production was behind schedule.

3. **Why** was production behind schedule?
  - Because one of the machines broke down.
4. **Why** did the machine break down?
  - Because it wasn't maintained properly.
5. **Why** wasn't it maintained properly?
  - Because there is no preventive maintenance schedule in place.

**Root Cause:** Lack of a preventive maintenance schedule.

#### **Benefits**

- **Simplicity** : Easy to use without requiring complex tools.
- **Focus on Root Causes**: Prevents "quick fixes" by targeting underlying issues.
- **Team Collaboration**: Encourages cross-functional discussion for better insights.

### **3.4 TEST METRICS, QA TEAM**

#### **3.4.1 Subject to Change by Organization**

The structure, components, and usage of test matrices vary depending on:

1. **Development Methodology:**

- **Agile**: Focuses on sprints, user stories, and iterative updates in the test matrix.
- **Waterfall**: More static, with an emphasis on end-to-end traceability.

2. **Organization Size:**

- **Startups**: May use simplified matrices focusing on key features and high-risk areas.
- **Enterprises**: Often have detailed matrices with multiple layers for compliance and audit purposes.

3. **Industry Standards:**

- **Regulated Sectors** (e.g., healthcare, finance): Include columns for compliance checks and audit trails.
- **Tech Companies**: Focus more on performance, scalability, and functional coverage.

4. **Tooling:**

- Organizations using tools like Jira with Zephyr Squad may dynamically link test cases to requirements, automatically updating the matrix.
- Smaller teams might rely on spreadsheets for manual tracking.

### 3.4.2 Resource Utilization

**Efficiently managing resources means:**

- **People:** Assign tasks based on each team member's skills and expertise.
- **Tools:** Leverage tools like Jira for tracking and Selenium for automation to streamline work.
- **Time:** Prioritize critical tests and automate repetitive tasks to improve efficiency.

**Why it's important:**

- Saves time.
- Reduces costs.
- Ensures high-quality product delivery.

### 3.4.3 Productivity in Software Testing:

#### 1. Efficient Test Execution:

- Testers need to execute test cases within a given time frame without compromising the quality of testing. Productivity in testing means running more tests, identifying bugs, and ensuring coverage without wasting time on unnecessary tasks.

#### 2. Effective Test Automation:

- Automation helps increase productivity in testing by reducing the time spent on repetitive tasks. Automated tests can be run quickly and repeatedly, improving test coverage and reducing human effort. This allows testers to focus on more complex scenarios and improve efficiency.

#### 3. Test Planning and Management:

- Proper planning ensures that tests are aligned with project timelines and goals, allowing testers to focus on the most critical areas. Tools like Jira (with Zephyr Squad) help manage test cases, track progress, and optimize resources, which boosts productivity.

#### 4. Bug Detection and Reporting:

- Quickly identifying and reporting defects increases productivity. The faster a bug is discovered and logged, the quicker the development team can fix it, ensuring that testing continues smoothly without delays.

#### 5. Collaboration and Communication:

- In testing, collaboration with developers, product managers, and other team members enhances productivity. Clear communication helps avoid redundant work, clarify requirements, and ensure alignment across teams, making the testing process more efficient.

#### 6. Resource Utilization:

- Properly allocating resources (testers, tools, environments) based on their skillset and the complexity of the tasks ensures higher productivity. For example, assigning automation tasks to skilled testers and manual testing to those familiar with the application.

## 7. Balancing Quality with Speed:

- Being productive in testing doesn't mean rushing through tests; it's about striking a balance between speed and quality. While tests must be completed on time, they must also be thorough and accurate to avoid missing critical defects.

### Benefits of Productivity in Testing:

- **Faster Releases:** Increased productivity leads to quicker test cycles, reducing the overall time to market.
- **Cost-Efficiency:** More tests are completed with fewer resources, lowering testing costs.
- **Better Quality:** Efficient testing ensures higher-quality products by identifying and fixing issues earlier in the development cycle.

### 3.4.4 Capacity Planning

Capacity Planning in Agile refers to the process of determining how much work a team can handle during a sprint or iteration, based on their available time, resources, and past performance. It helps teams balance the workload effectively, ensuring they commit to a realistic number of user stories or tasks without overloading themselves.

1. **Team Size:** There are 5 members in the team.
2. **Working Time:** Each team member works for 8 hours per day.
3. **Daily Effort:** The total daily effort for the team is 40 hours (5 members \* 8 hours).
4. **Sprint Days:** A sprint lasts for 20 days (equivalent to 4 weeks with 5 working days each).
5. **Total Effort per Sprint:** Assuming each team member works for 8 hours per day, the total effort for one sprint is calculated as follows:

Total Effort=5 team members×8 hours/day×20 days=800 hours

Total Effort=5 team members×8 hours/day×20 days=800 hours

6. **1 Story Point:** Each story point represents 8 hours of effort.
7. **Total Story Points per Sprint:** Since the total effort for one sprint is 800 hours and each story point represents 8 hours,

The total number of story points for one sprint is calculated as follows:

Total story points for the sprint = Total effort for one sprint / Effort per story point

Total story points for the sprint = 800 hours / 8 hours/story point

Total story points for the sprint = 100 story points

**Total Points=800 hours/(8 hours/point)=100 points**

- 8. Sprint Meetings:** Assuming 20% of the sprint capacity is reserved for meetings, the total points that can be considered for work in one sprint is calculated as follows:

Total Points for Work=100 points–(20%×100 points) =100 points–20 points=80 points

### **3.4.5 QA Team Structure , Different Roles and Responsibilities**

➤ **QA Manager/Director:**

- Responsible for overseeing the entire QA process.
- Sets QA goals, strategies, and priorities aligned with the organization's objectives.
- Manages resources, budgets, and timelines for QA activities.
- Acts as a liaison between QA team and other departments, such as development and product management.

➤ **QA Leads/Team Leads:**

- Lead smaller groups or teams within the QA department.
- Coordinate daily activities, assign tasks, and monitor progress.
- Provide guidance, support, and mentorship to QA engineers.
- Report to the QA manager/director on team performance and project status.

➤ **QA Engineers/Testers:**

- Responsible for executing testing activities to ensure software quality.
- Develop test plans, test cases, and test scripts based on project requirements.
- Conduct various types of testing, including functional, regression, integration, and performance testing.
- Report defects, track their resolution, and verify fixes.
- Collaborate with developers, product managers, and other stakeholders to identify and address quality issues.

➤ **Automation Engineers:**

- Develop and maintain automated test scripts and frameworks.
- Identify opportunities for test automation and implement automated testing solutions to improve efficiency and coverage.
- Collaborate with QA engineers to integrate automated tests into the continuous integration/continuous deployment (CI/CD) pipeline.
- Monitor and analyse automated test results, identify trends, and recommend improvements.