BCSE301L_SOFTWARE ENGINEERING

VALIDATION AND VERIFICATION

module 5

Presented By,
Dr.Baiju B V
Assistant Professor
School of Computer Science and Engineering VIT, Vellore

Strategic Approach to Software Testing

- Testing is a set of activities that can be planned in advance and conducted systematically.
- A strategy for software testing must accommodate
  - **low-level tests** that are necessary to verify that a small source code segment has been correctly implemented
  - **high-level tests** that validate major system functions against customer requirements.
- It is a set of guidelines that an ***internal QA department*** or an ***external QA team*** must adhere to in order to deliver the standard of quality you have established.

1. Verification and Validation

- **Verification** refers to the set of tasks that ***ensure that software correctly implements a specific function.***
- **Validation** refers to a different set of tasks ***that ensure that the software that has been built is traceable to customer requirements***.

Verification: *"Are we building the product right?"*
Validation: *"Are we building the right product?"*

VERIFICATION

- 2 sleeves?
- Is it size L?
- Is it blue?
- Are any buttons missing?

VALIDATION

- Does it fit?
- Is it comfortable to drive in?
- Does the colour match my eyes?
- Can I afford it?
- Is it good quality?
- Will my date like it?

Verification & Validation

Are we building the product right?

Are we building the right product?

Verification

- Verify the intermediary products like requirement documents, design documents, ER diagrams, test plan and traceability matrix
- Developer point of view
- Verified without executing the software code
- Techniques used: Informal Review, Inspection, Walkthrough, Technical and Peer review

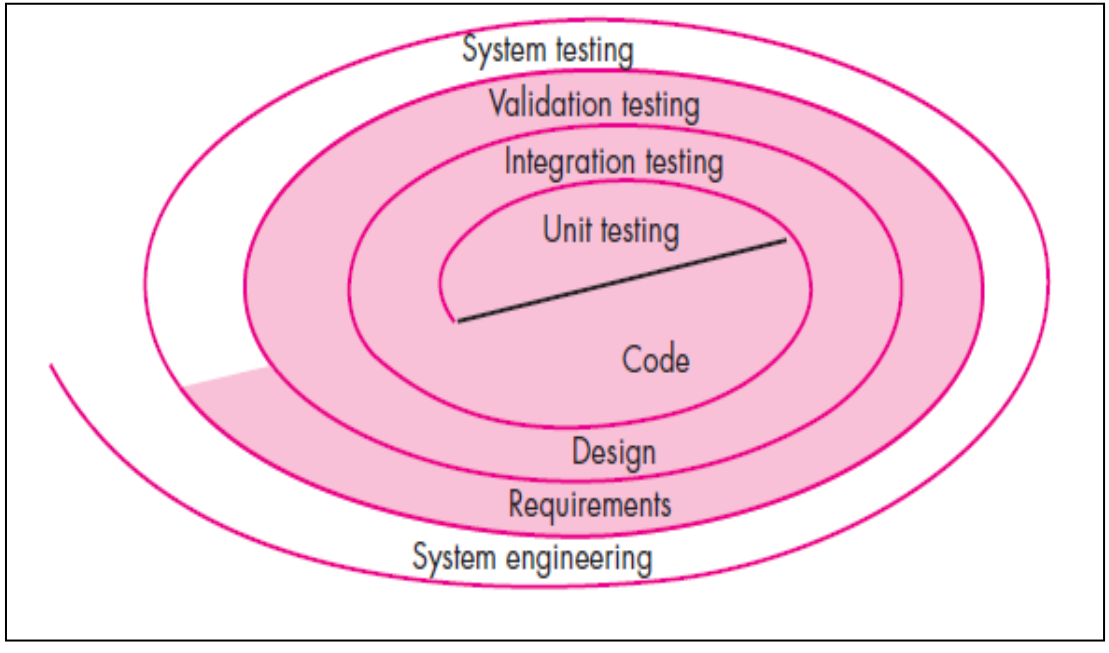
Validation

- Validate the final end product like developed software or service or system
- Customer point of view
- Validated by executing the software code
- Techniques used: Functional testing, System testing, Smoke testing, Regression testing and Many more

- Verification and validation includes a wide array of SQA activities:
  - *Technical reviews*
  - *Quality and configuration audits*
  - *Performance monitoring*
  - *Simulation*
  - *Feasibility study*
  - *Documentation review*
  - *Database review*
  - *Algorithm analysis*
  - *Development testing*
  - *Usability testing*
  - *Qualification testing*
  - *Acceptance testing*
  - *Installation testing.*

2. Software Testing Strategy

- The software process may be viewed as the spiral illustrated in Figure.
- System engineering first defines the role of software.
- Then, software requirements analysis identifies key aspects like
  - data
  - functions
  - behavior
  - performance
  - constraints
  - validation criteria



- Moving inward along the spiral, you come to design and finally to coding.
- To develop software, start with high-level ideas and gradually refine them, moving step by step toward detailed implementation.

- A strategy for software testing may also be viewed in the context of the spiral.
- **Unit testing** starts at the core of development, focusing on testing individual components like functions, classes, or modules to ensure they work correctly in the source code.
- Testing moves outward in stages, reaching **integration testing**, where the focus is on design and software architecture.
- **Validation testing** checks if the final software meets the initial requirements. It ensures that the software does what it was designed to do.
- Finally, in **system testing**, the entire software and its components are tested together to ensure they work as a complete system.
- To test software, you start small and gradually expand testing in a spiral pattern, covering more features with each step.

3. Strategic issues

- Tom Gilb says that a software testing strategy will succeed when software testers

(i) Specify product requirements in a quantifiable manner long before testing commences.

- A good testing strategy also assesses other quality characteristics such as portability, maintainability, and usability .
- These should be specified in a way that is measurable so that testing results are unambiguous.
- **The system should process 1,000 transactions per second with a response time of under 2 seconds.**

(ii) State testing objectives explicitly

- Specific objectives of testing should be stated in measurable terms.
- For example,
  - Test effectiveness (Identify at least 90% of critical defects before release)
  - Test coverage
  - Meantime- to-failure (System should operate for at least 1000 hours before encountering a failure)
  - Cost to find and fix defects

(iii) Understand the users of the software and develop a profile for each user category.

– Use cases that describe the interaction scenario for each class of user can reduce overall testing effort by focusing testing on actual use of the product

(iv) Develop a testing plan that emphasizes "rapid cycle testing."

– Recommends that a software team "learn to test in rapid cycles (2 percent of project effort) of customer-useful, at least field 'trialable,' increments of functionalityand/or quality improvement."

(v) Build "robust" software that is designed to test itself.

– Software should be designed in a manner that uses antibugging techniques.

(vi) Use effective technical reviews as a filter prior to testing.

– Technical reviews can be as effective as testing in uncovering errors.

(vii) Conduct technical reviews to assess the test strategy and test cases themselves.

– Technical reviews can uncover inconsistencies, omissions, and outright errors in the testing approach. This saves time and also improves product quality.

Testing Fundamentals

- The goal of testing is to find errors, and a good test is one that has a high probability of finding an error.

(i) A good test has a high probability of finding an error.

  - To achieve this goal, the tester must understand the software and imagine possible ways it might fail.

(ii) A good test is not redundant.

  - Testing time and resources are limited.
  - There is no point in conducting a test that has the same purpose as another test.
  - Every test should have a different purpose.

(iii) A good test should be "best of breed" .

  - When time and resources are limited, only the tests most likely to reveal major errors should be run.

Redundant Testing (Inefficient)

Test Case 1: Verify that a user can add an item to the shopping cart.

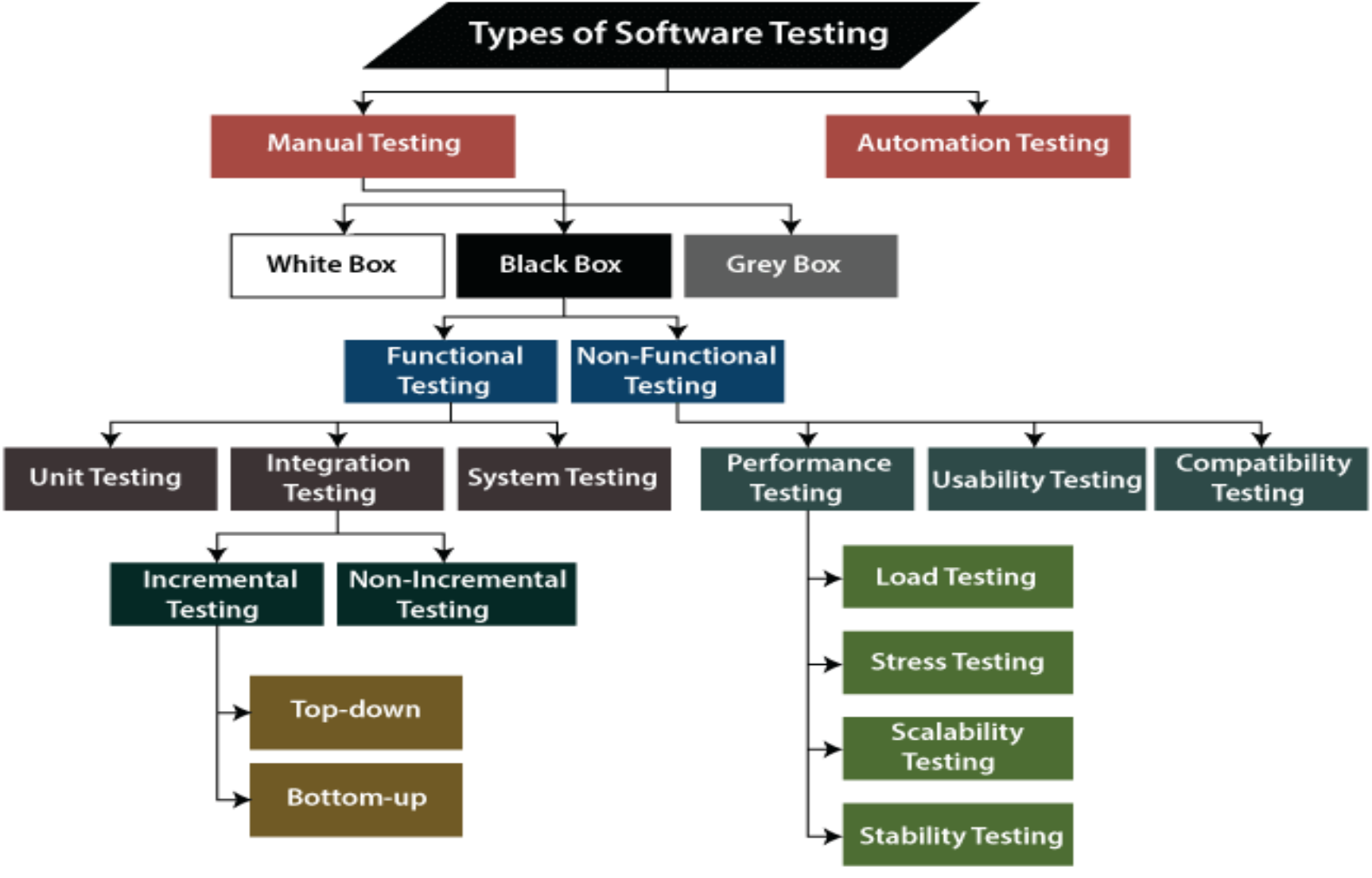
Test Case 2: Verify that clicking the "Add to Cart" button adds the item to the cart.

Efficient Testing (Non-Redundant)

Test Case 1: Verify that a user can add an item to the cart and the cart updates correctly.

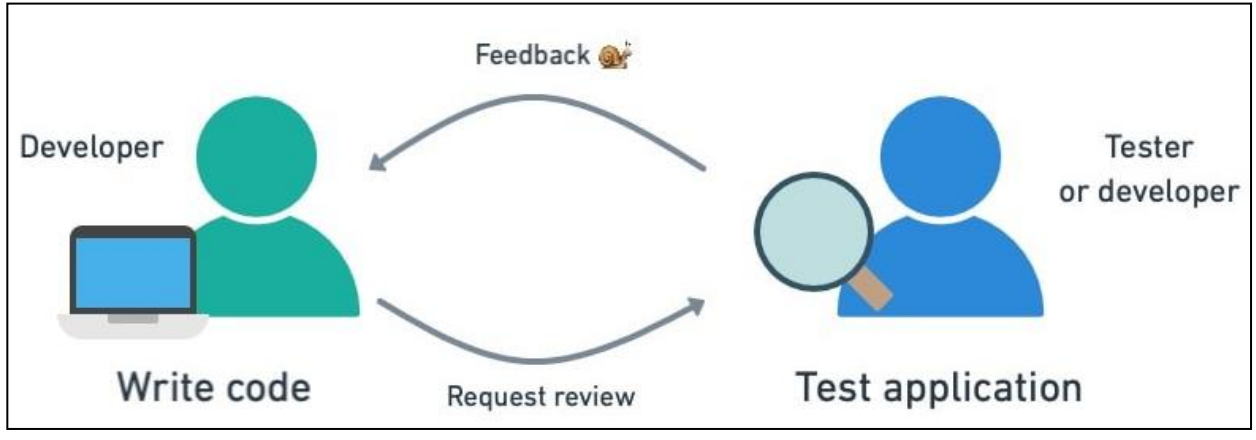
Test Case 2: Verify that the system correctly calculates the total price after adding multiple items.

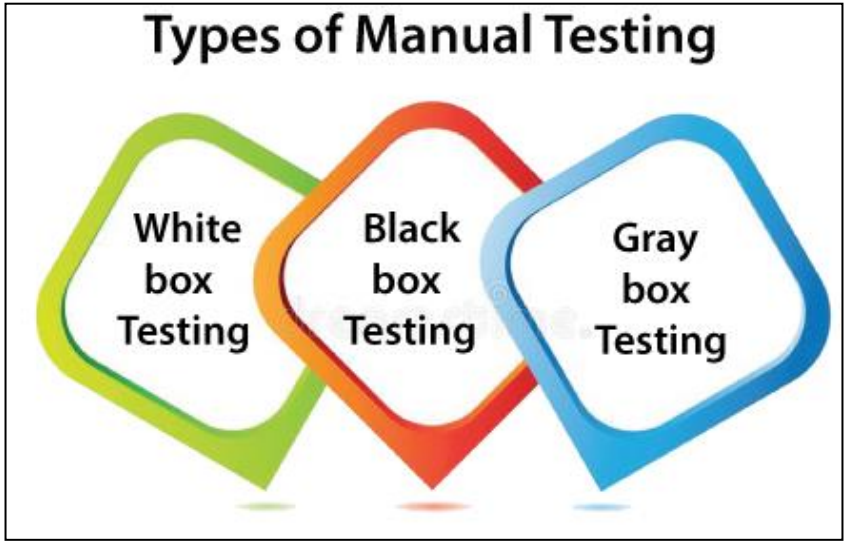
Test Case 3: Verify that a user can successfully remove an item from the cart.

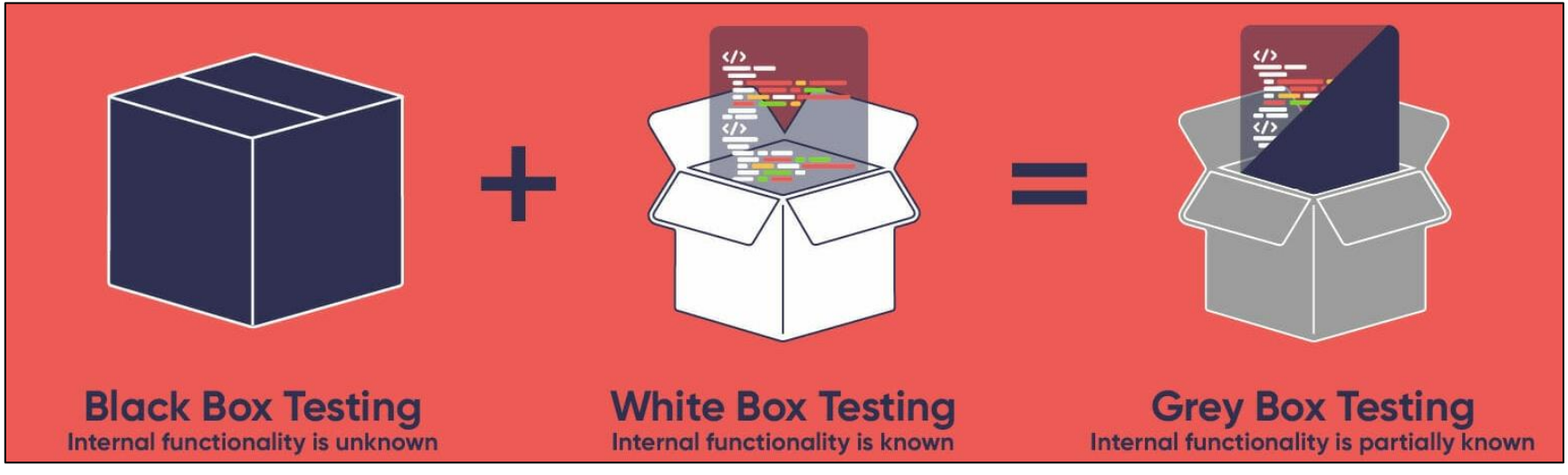

Types of Software Testing
Manual Testing
Automation Testing
White Box
Black Box
Grey Box
Functional Testing
Non-Functional Testing
Unit Testing
Integration Testing
System Testing
Performance Testing
Usability Testing
Compatibility Testing
Incremental Testing
Non-Incremental Testing
Top-down
Bottom-up
Load Testing
Stress Testing
Scalability Testing
Stability Testing

1. Manual Testing

- Manual Testing is a type of software testing in which ***test cases are executed manually by a tester*** without using any automated tools.
- The purpose of Manual Testing is to ***identify the bugs***, ***issues***, and ***defects*** in the software application.
- Manual Testing is one of the most fundamental testing processes as it can ***find both visible and hidden defects of the software***.
- Manual testing is mandatory for every newly developed software before automated testing.

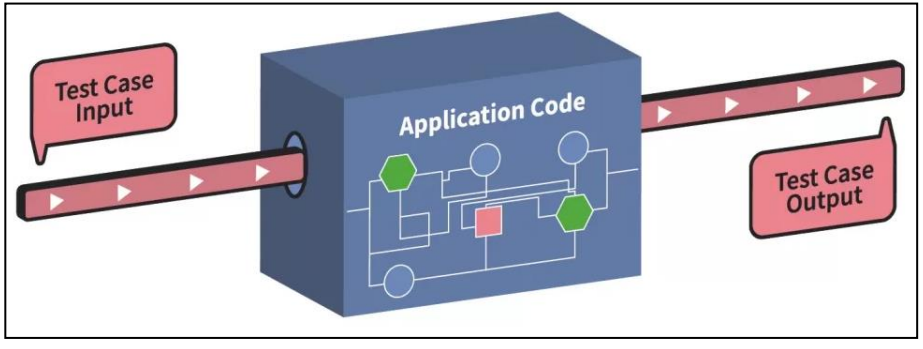



Types of Manual Testing
White box Testing
Black box Testing
Gray box Testing


Black Box Testing
Internal functionality is unknown
White Box Testing
Internal functionality is known
Grey Box Testing
Internal functionality is partially known

A. White Box Testing

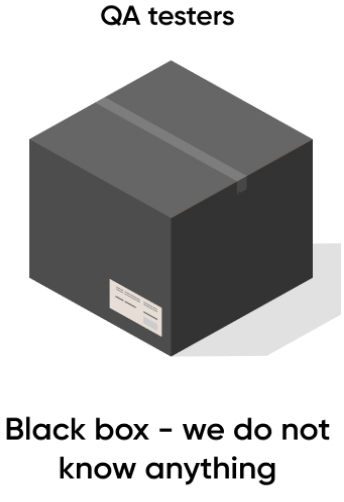
- White box testing checks the ***internal structure***, ***code***, and ***logic*** of a software program to find errors.
- It ensures that ***data flows correctly***, all ***conditions are tested***, and every part of the code works as expected.
- It is also called ***glass box testing*** or ***clear box testing*** or ***structural testing.***

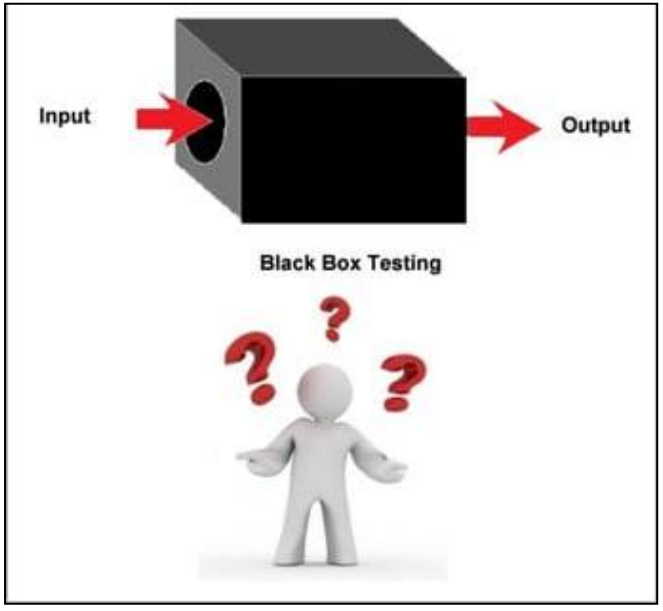


- White Box Testing focus on
  - ***Verification of security holes in source code***
  - ***Testing of any broken or incomplete path***
  - ***To verify the flow of structure as mentioned in the software requirement document.***
  - ***To check if all the expected outcomes are met***
  - ***Line-by-line verification of code***

B. Black-Box Testing

- Black box testing is a technique of software testing which ***examines the functionality of software*** without peering into its internal structure or coding.
- The primary source of black box testing is a ***specification of requirements*** that is stated by the customer.
- Tester just interacts with the application and tests the functional and non-functional behavior of the application.
- Tester selects a function and gives input value to examine its functionality, and checks whether the function is giving expected output or not.
- If the function produces correct output, then it is passed in testing, otherwise failed.





Let's take an example of black-box testing on the login page. Test cases will be based on how the end-user interacts with the login page. Some of them are:

- ***Users should be able to login with a valid ID and Password.***
- ***Users should not log in with an invalid ID and Password, and the software should show them the correct error message.***
- ***Border conditions like blank ID or Password checked***
- ***Other links on the page like Remember me, Forgot password should take the user to the right pages.***
- ***Here we are not concerned with how the login code works on the backend.***

Test Case	Input (Username & Password)	Expected Output	Result
Valid Login	user123 / Pass@123	Login Successful	Pass/Fail
Invalid Password	user123 / WrongPass	Incorrect Password Message	Pass/Fail
Invalid Username	wrongUser / Pass@123	User Not Found Message	Pass/Fail
Empty Fields	"" / ""	Error: Fields cannot be empty	Pass/Fail
SQL Injection Attempt	' OR 1=1; -- / anything	Error: Invalid Credentials	Pass/Fail

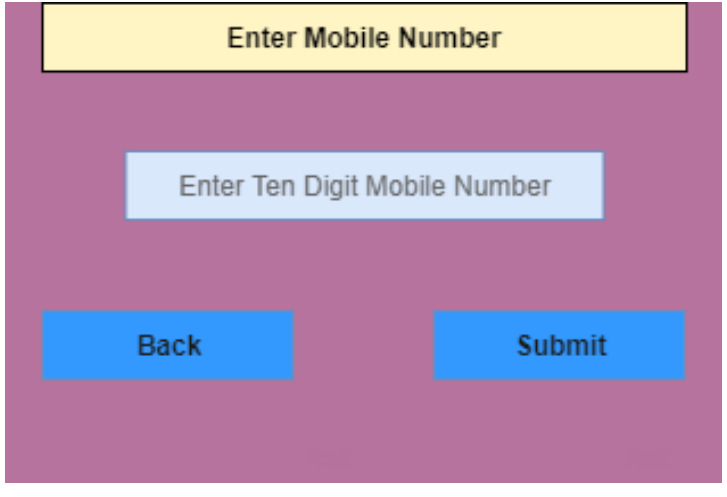
Black Box Testing Techniques

1. Equivalence Partitioning

- This technique, also called equivalence class partitioning, is used to ***divide the input data into groups (partitions) of valid and invalid values***.
- The grouping needs to be such that either all values in a set are valid or invalid.
- It is used to ***minimize the number of possible test cases*** to an optimum level while maintains reasonable test coverage.

A function of the software application accepts a 10 digit mobile number.

INVALID 1 Test case	INVALID 2 Test case	VALID 3 Test case	VALID
DIGITS >=11	DIGITS<=9	DIGITS = 10	DIGITS =10
93847262219	984543985	9991456234	9893451483



The discount is calculated depending on the total amount of the shopping cart.

- ***If the total amount is in the range of $100–$200, the discount is 10%.***
- ***If the total amount is in the range of $201–$500, the discount is 20%.***
- ***If the total amount is more than $500, the discount is 30%.***

In this scenario, we can identify three valid partitions and one invalid partition for the amount under $100.

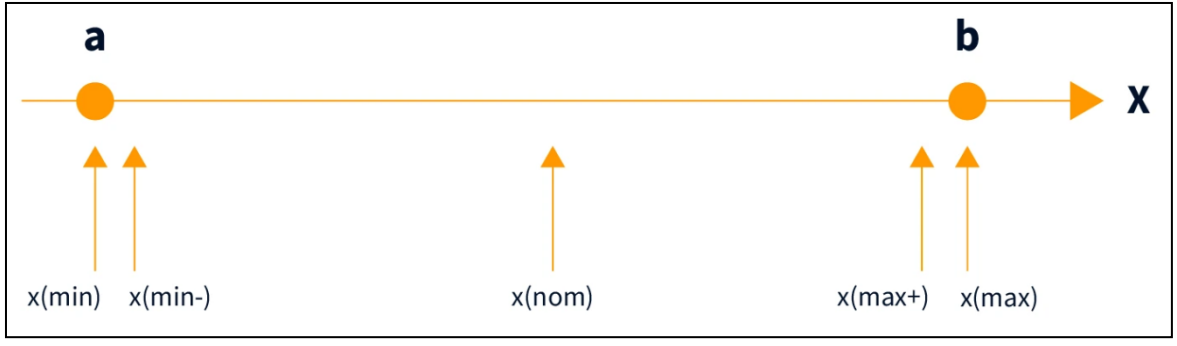
we can take one value from each partition:

- ***$140 from the first valid partition***
- ***$370 from the second valid partition***
- ***$530 from the third valid partition***
- ***$78 from the invalid partition.***

Now we have four test cases and have achieved 100% coverage because all defined partitions are covered.

2. Boundary Value Analysis

- In this technique, the behavior of the software at the ***input boundaries is tested***.
- It is common to include a value from the middle of the input range.
- Boundary values are those that contain the ***upper and lower limit of a variable***.
- It is most suitable for the systems where an input is within certain ranges.



Consider a software application that requires users to enter the age. Applications may contain specific minimum and maximum age restrictions. The restriction is to enter the age between 18 to 30 years. If the user enters values above 30 or below 18, the application may not work as expected

Invalid case	Valid case	Invalid case
11,12,13,14,15,16,17	21,22,23,24,25,26,27,28,29	31,32,33,34,35,36,37,38,39

3. Decision Table Testing

- This method ***captures different input combinations*** and their ***expected results in a tabular form and design test cases based on this table***.

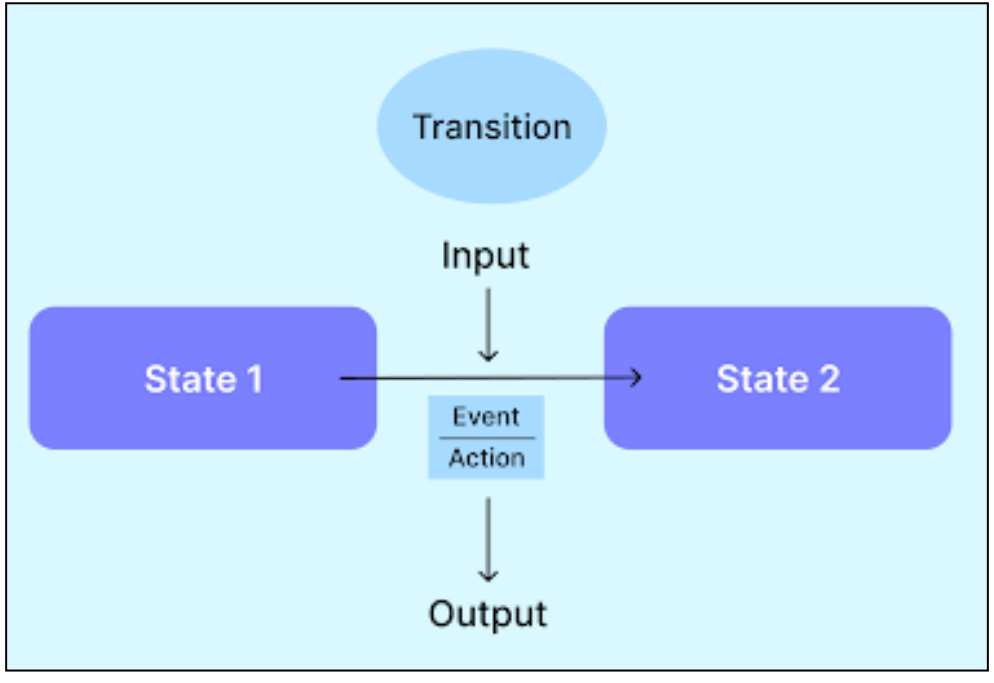
Most of us use an email account, and when you want to use an email account, for this you need to enter the email and its associated password.

If both email and password are correctly matched, the user will be directed to the email account's homepage; otherwise, it will come back to the login page with an error message specified with "Incorrect Email" or "Incorrect Password."

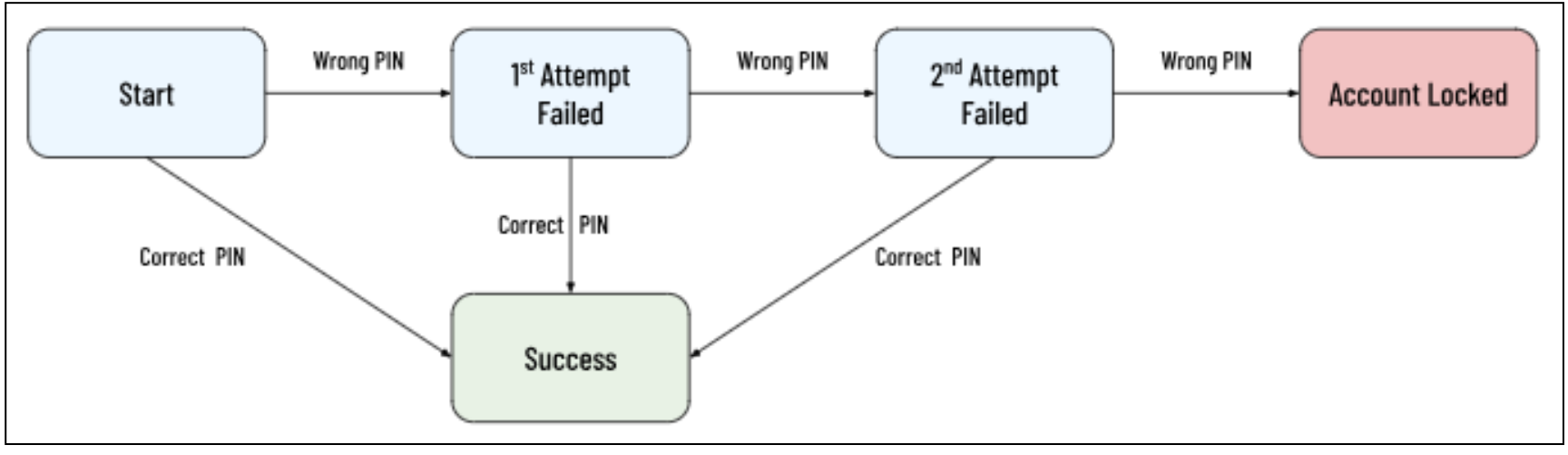
Email (condition1)	T	T	F	F
Password (condition2)	T	F	T	F
Expected Result (Action)	Account Page	Incorrect password	Incorrect email	Incorrect email

4. State Transition Testing

- This technique is used when the ***software behavior depends on past values of inputs***.
- The software is considered to have a finite number of states.
- The transition from one state or another of **Application Under Test** (AUT) happens in the responses to the action of the users.
- State transition testing helps understand the system's behavior and covers all the conditions.
- The four main components for a state transition diagram are as follows:
  - States
  - Transition
  - Events
  - Actions



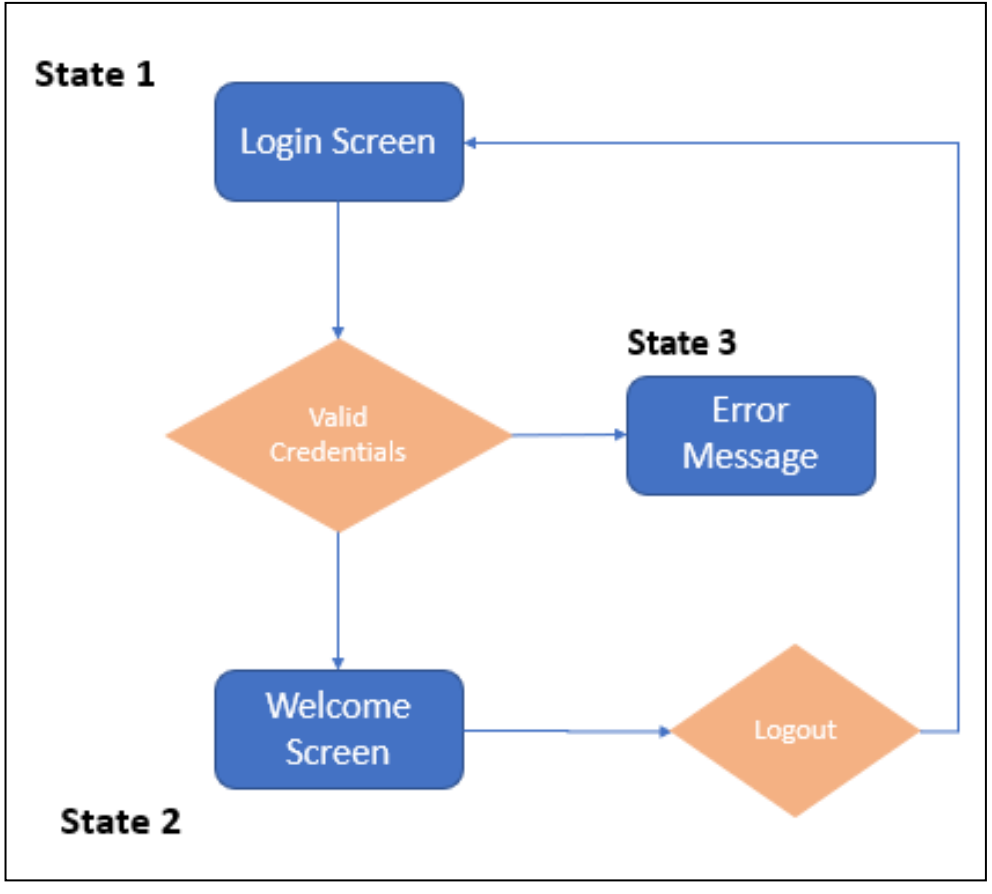
Consider a bank application that allows users to log in with valid credentials. But, if the user doesn't remember the credentials, the application allows them to retry with up to three attempts. If they provide valid credentials within those three attempts, it will lead to a successful login. In case of three unsuccessful attempts, the application will have to block the account.



Example : Let take a simple example of an application with just a login functionality.

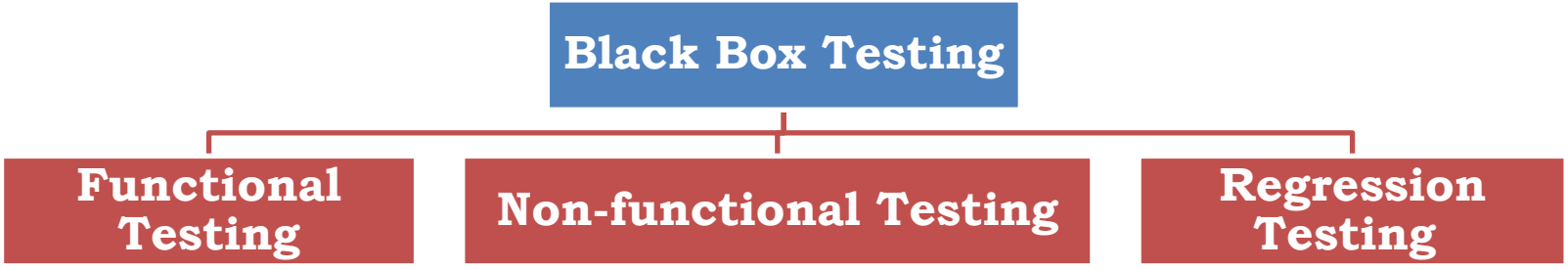
Rule:

User (assuming an existing user) can only login with valid credentials and should get a welcome message. With invalid credentials, the user should get an error message.



Types of Black Box Testing

- There are three types of black-box testing namely



(i) Functional Testing

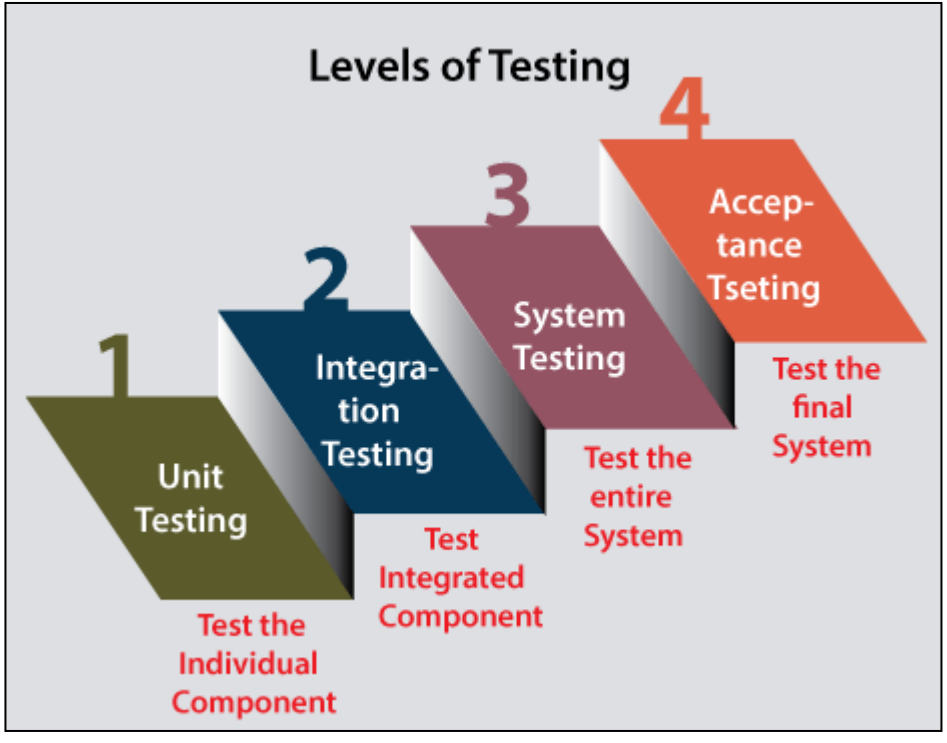
- Functional testing is a type of testing that ***validates the end-user features of the software***.
- It ensures that the software's characteristics ***match the end-user requirements*** and the ***software is bug-free***.
- Functional testing can be done either manually or using automation.
- Principal usability tests are performed, including checks to verify whether:
  - ***UI elements like text and images are correctly displayed***
  - ***Users can navigate through the screens without a problem***
  - ***The application is responsive on different devices***
  - ***The software is usable by people with disabilities and other disadvantaged groups***

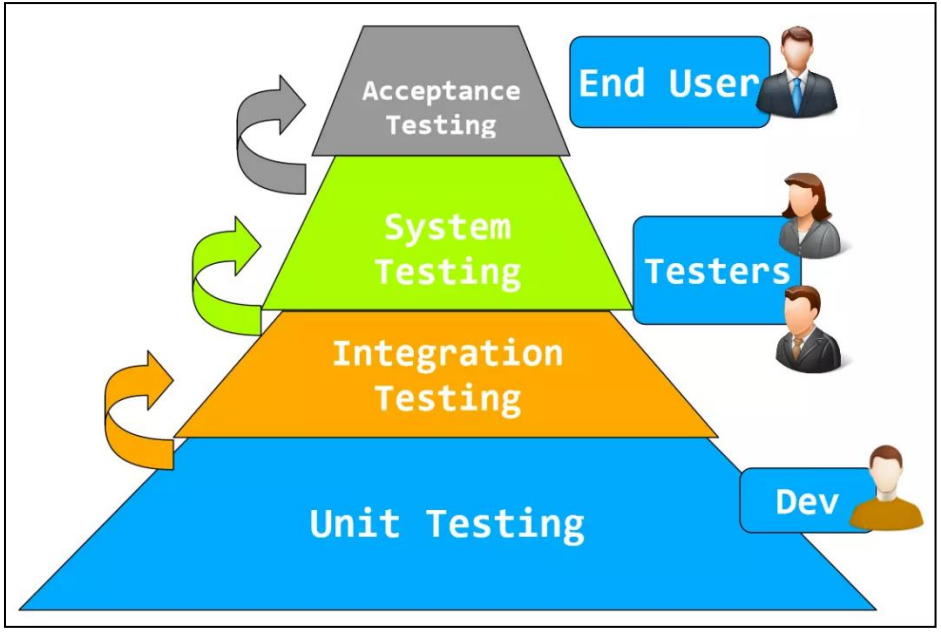
(ii) Non Functional Testing

- Non-Functional Testing is defined as a type of Software testing to ***check non-functional aspects*** (performance, usability, reliability, etc) of a software application.
- It is designed to test the readiness of a system as per nonfunctional parameters which are never addressed by functional testing.
- Non-functional testing is essential to ensure that the system can handle the load and perform as expected under real-world conditions.

Levels of Testing

- Testing levels are the procedure for finding the missing areas and avoiding overlapping and repetition between the development life cycle stages.





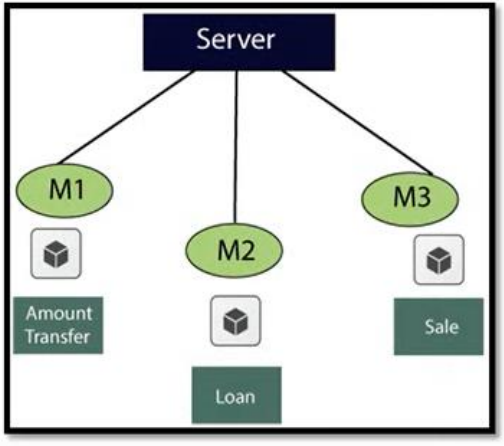
1. Unit testing

- Unit testing is testing the ***smallest testable unit of an application***.
- In a testing level hierarchy, unit testing is the first level of testing done before integration and other remaining levels of the testing.
- Unit testing ***fixes defects very early in the development phase*** that's why there is a possibility to occur a smaller number of defects in upcoming testing levels.

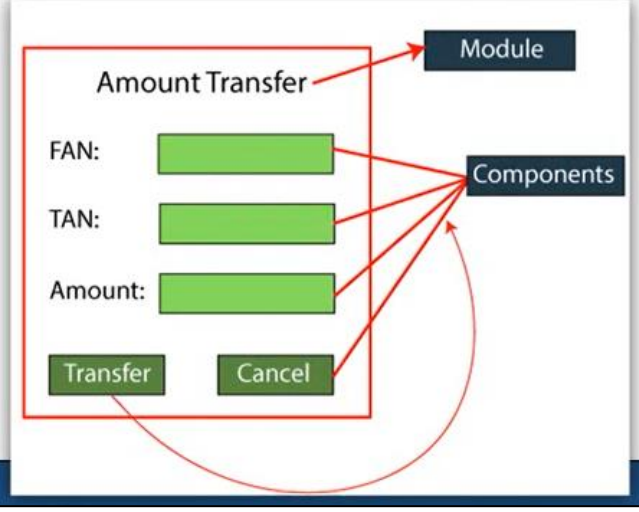
A restaurant needs an app that helps customers order at their tables without a server. The developer would create a unit test to examine the "add to order" function. Other individual functions such as "remove from order" or "submit order" would also go under unit testing.

Suppose a car manufacturer is making a car. He will make different parts like its engine, ignition system, wheels, etc. These different parts are the units that he will test individually. The manufacturer will check if the ignition system and the engine are working fine or not.

Example of Unit Testing



For From Account Number (FAN) & To Account Number (TAN)	
Values	**Description**
1234	Account Number Accepted
4300	Error: Account Number Invalid
Blank	Error: Enter Account Number
Alphanumeric	Error: Enter digits only
Block Acc. No	Error Message



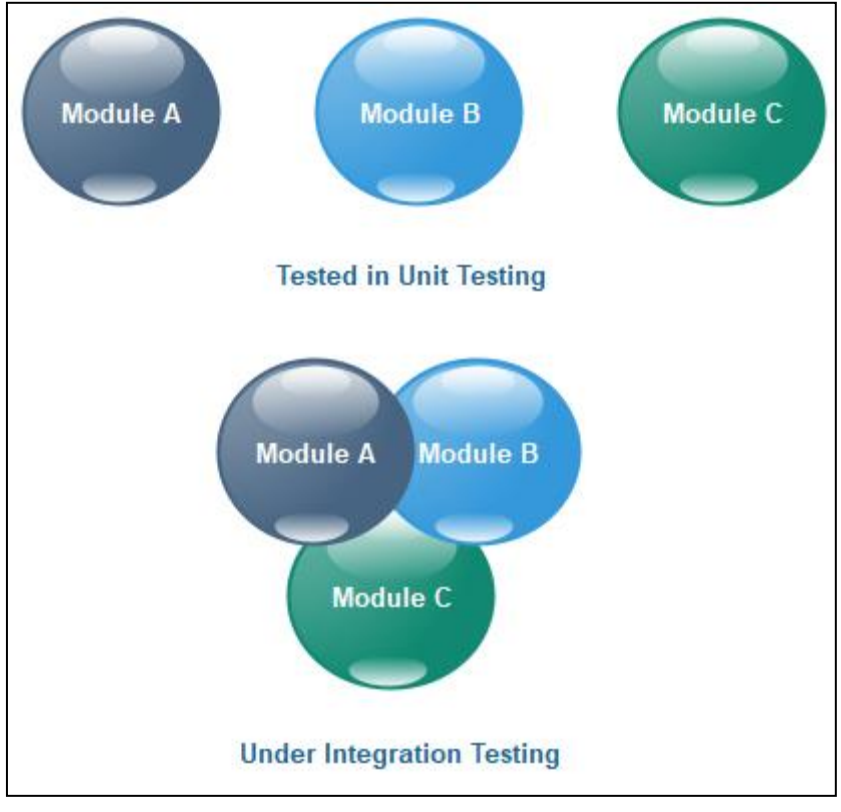
For Transfer Button
Description
Enter valid FAN value
Enter valid TAN value
Enter correct amount
Click on Transfer Button

For Cancel Button
Description
Enter value of FAN, TAN & Amount
Click on Cancel Button: All data should be clear

2. Integration testing

- Integration testing is a software testing technique that involves ***testing the integration of different software components*** to ensure that they work together as expected.

- Integration testing aims to ***detect any defects in the interaction between the integrated components.***
- Once all the modules have been unit-tested, integration testing is performed.
- Integration testing aims to ***evaluate the accuracy of communication*** among all the modules.



3. System Testing

- System testing verifies whether the ***complete software product to be delivered meets the specifications described in the requirement document***.
- The software is developed in units and then interfaced with other software and hardware to create a complete computer system.

Suppose a car-making company has already made systems like the ignition system, braking system, engine, fuel system, accelerator, GPS, air conditioner, etc., which were tested individually (Unit testing).

The collaboration of these systems is tested(integration testing) for example the fuel system is checked in collaboration with the car engine. But when all systems are ready, the company will check all the systems working by actually driving the car to check if all the systems are working fine with each other. That will be system testing.

- System Testing is performed in the following steps:

Test Environment Setup: Create testing environment for the better quality testing.

Create Test Case: Generate test case for the testing process.

Create Test Data: Generate the data that is to be tested.

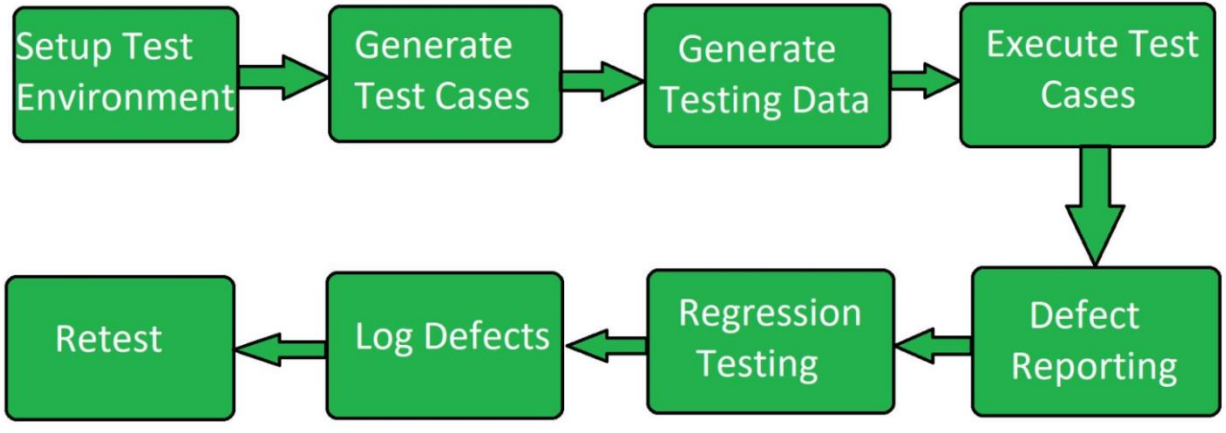
Execute Test Case: After the generation of the test case and the test data, test cases are executed.

Defect Reporting: Defects in the system are detected.

Regression Testing: It is carried out to test the side effects of the testing process.

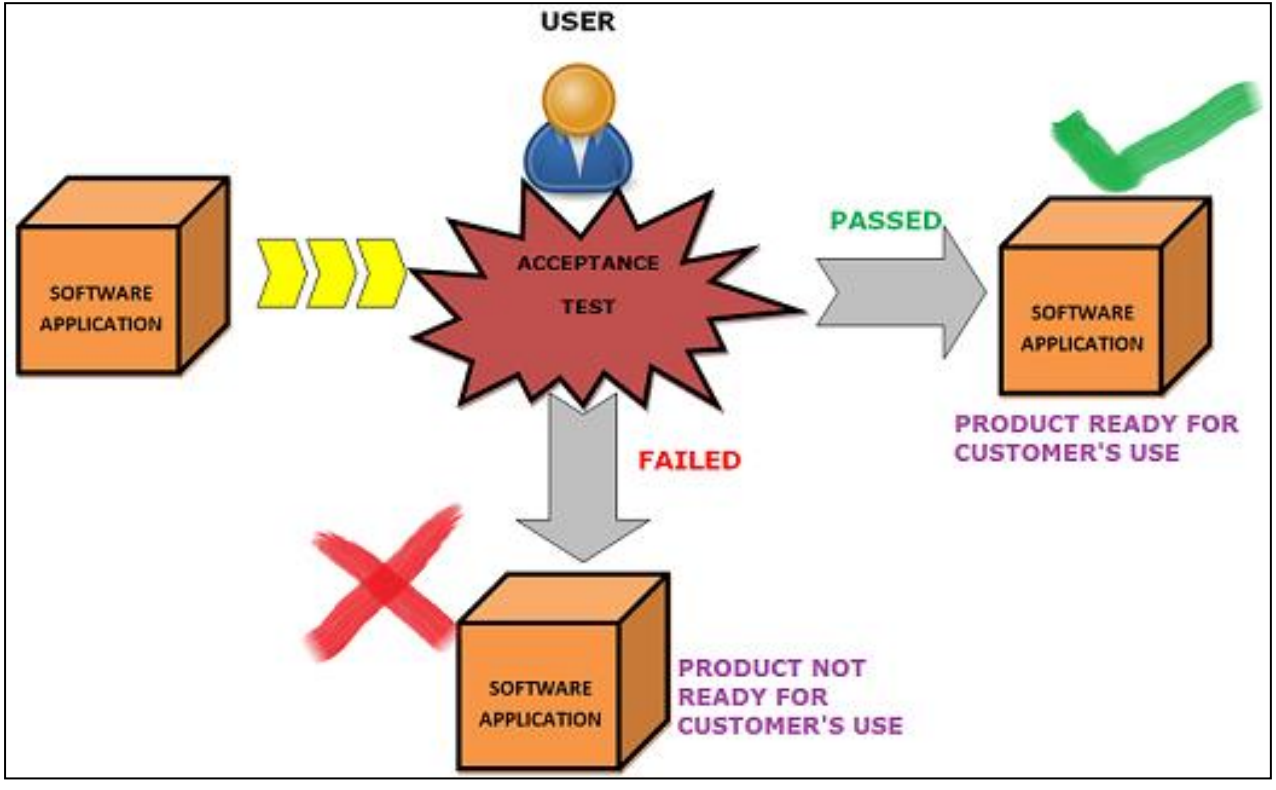
Log Defects: Defects are fixed in this step.

Retest: If the test is not successful then again test is performed.

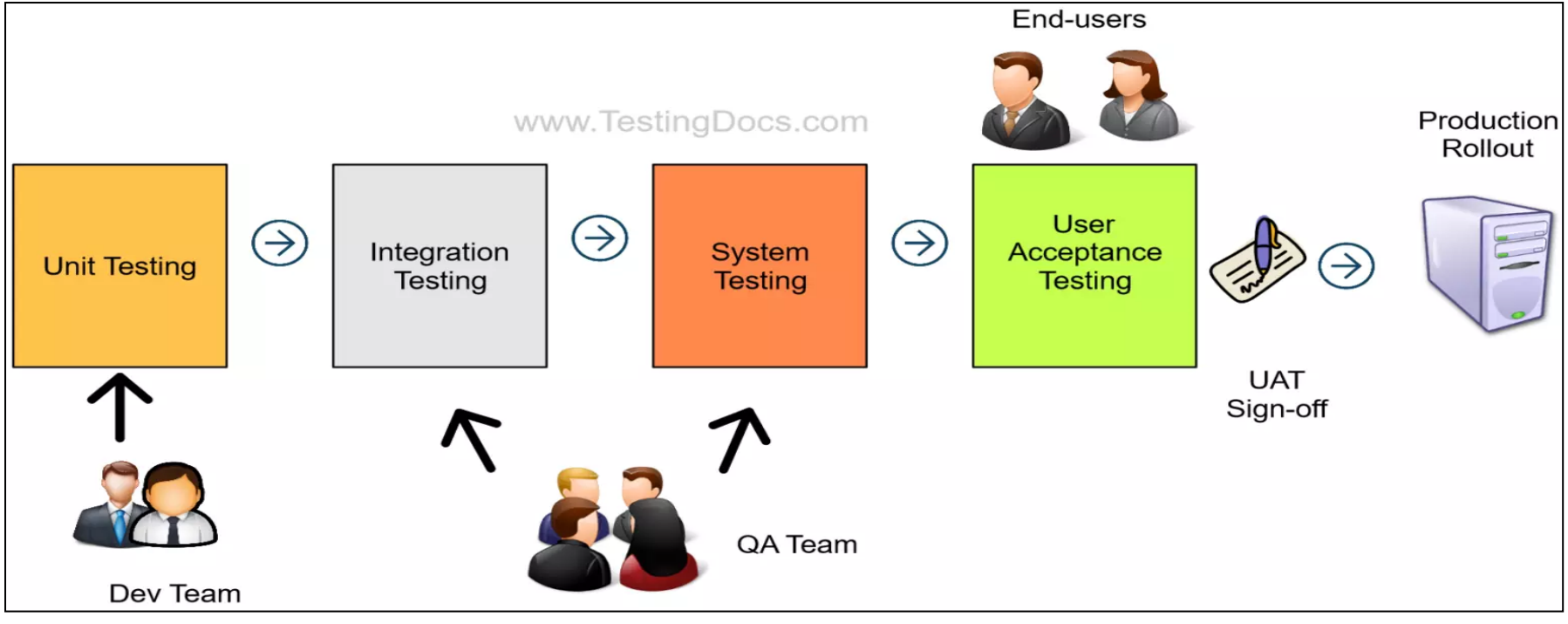


4. Acceptance Testing

- Acceptance testing is ***formal testing*** based on user requirements and function processing.
- It determines whether the software is conforming specified requirements and user requirements or not.

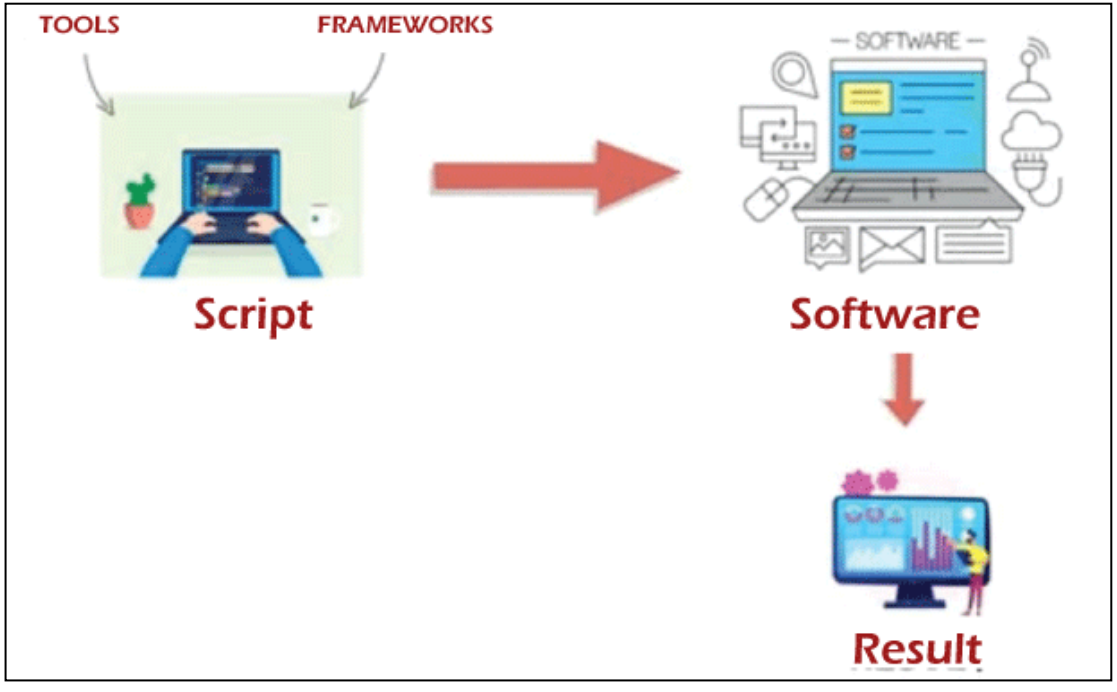


Consider the manufacturing of a chair. Wood, steel, plastic, paints, are the items needed for its manufacture. These units are separately produced and unit tested. After that, these units are combined together, and integration testing is performed. After the integration process, system testing is performed. System testing is followed by acceptance testing where the final product is matched with the requirements of the end-users.



2. Automation Testing

- Automation testing refers to the automatic testing of the software in which developer or tester ***write the test script*** once with the help of testing tools and framework and run it on the software.
- The test script automatically test the software without human intervention and shows the result (either error, bugs are present or software is free from them)."



Regression Testing

- **Regression Testing** is a type of software testing that checks if recent code changes have affected existing features.
- It ensures that the software still works properly after modifications.
- This testing is done when there are significant changes in the code to confirm that everything functions as expected.
- Regression tests are also known as the **Verification Method**.
- Here are the scenarios where the regression testing process can be applied:

  (i) New functionality is added to the application

  – This happens when new features or modules are created in an app or a website.

  – The regression is performed to see if the existing features are working as usual with the introduction of the new feature.

A website has a login functionality which allows users to log in only with Email. Now providing a new feature to do login using Facebook.

(ii) In case of change requirement:

- When any significant change occurs in the system, regression testing is used.
- This test is done to check if these shifts have affected features that were there.

Remember password removed from the login page which is applicable previously.

(iv) After a defect is fixed:

- The developers perform regression testing after fixing a bug in any functionality.
- This is done to determine if the changes made while fixing the bug have affected other related existing features.

Assume login button is not working in a login page and a tester reports a bug stating that the login button is broken. Once the bug fixed by developers, tester tests it to make sure Login Button is working as per the expected result. Simultaneously, tester tests other functionality which is related to the login button.

(v) Once the performance issue is fixed:

– After fixing any performance issues, the regression testing process is triggered to see if it has affected other existing functional tests.

Loading of a home page takes 5 seconds, reducing the load time to 2 seconds.

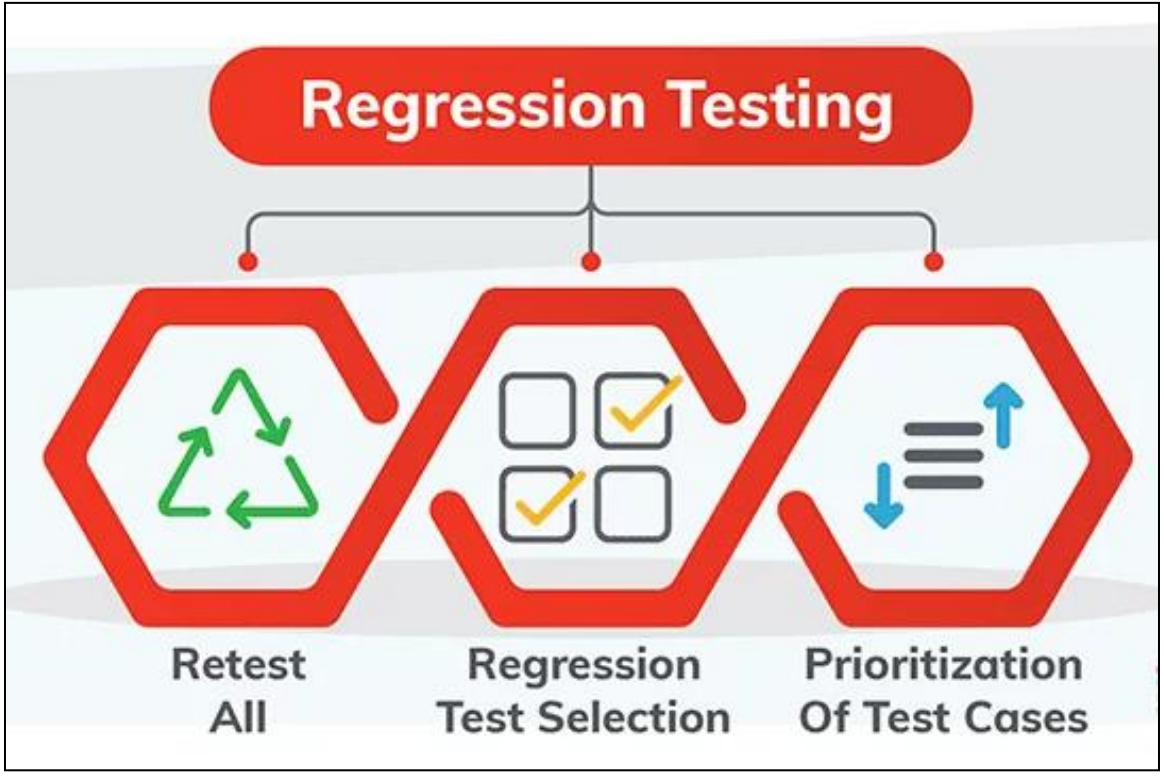
(vi) While integrating with a new external system:

– End-to-end regression testing process is required whenever the product integrates with a new external system.

When we update the database from MySql to Oracle.

How to perform Regression Testing?

- The need for regression testing comes when software maintenance includes ***enhancements***, ***error corrections***, ***optimization***, and ***deletion of existing features***.
- Regression testing can be performed using the following techniques:



1. Re-test All:

- This is the technique where ***test engineers*** **execute all the existing test cases** without any miss.
- This is quite ***expensive as it requires huge time and resources***.

2. Regression Test Selection

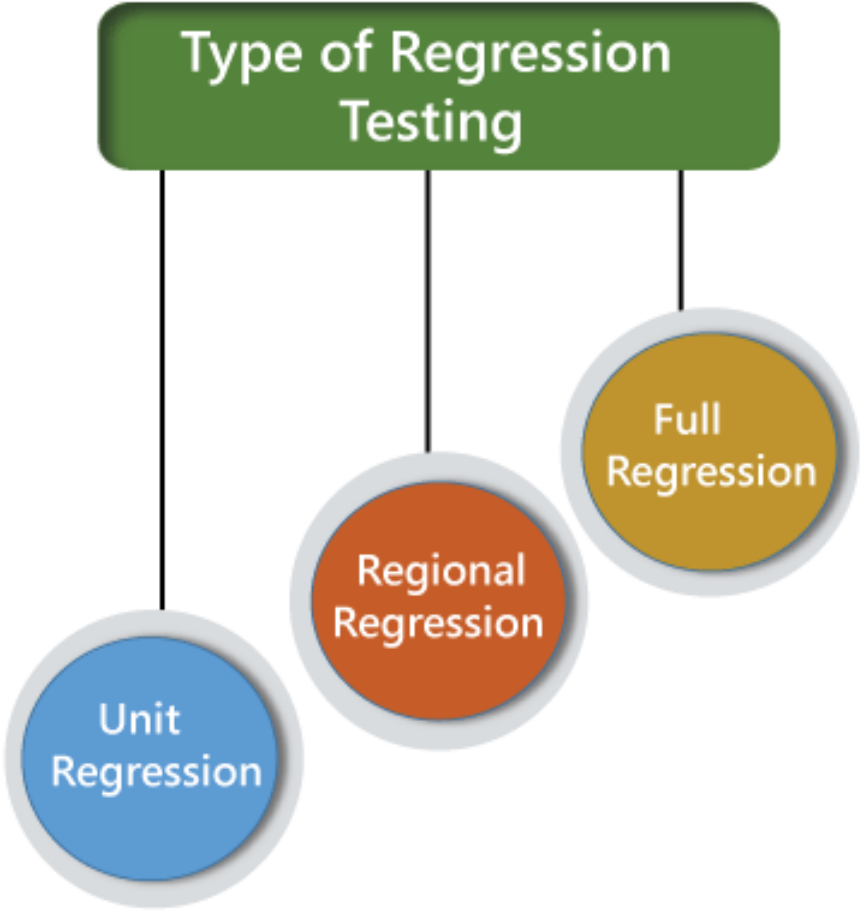
- In this technique, test engineers **select a subset of test cases** based on the impact analysis.
- Test chosen cases categorized as
  - ***Reusable Test cases***
  - ***Obsolete Test cases***
- Reusable Test cases used in future regression cycles.
- Obsolete Test cases no longer used in future cycles.

3. Prioritization of test cases:

- Prioritize the test case **depending on business impact**, **critical and frequently functionality used**.
- Selection of test cases will reduce the regression test suite.

Types of Regression Testing

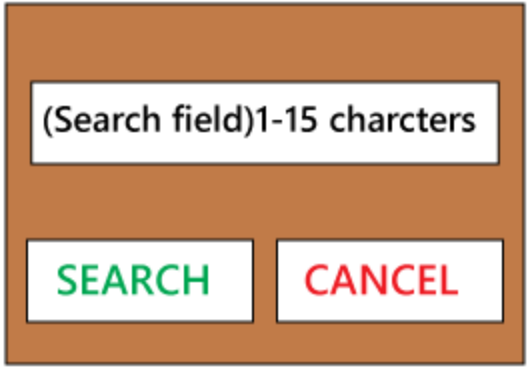
- The different types of Regression Testing are as follows:



1) Unit Regression Testing [URT]

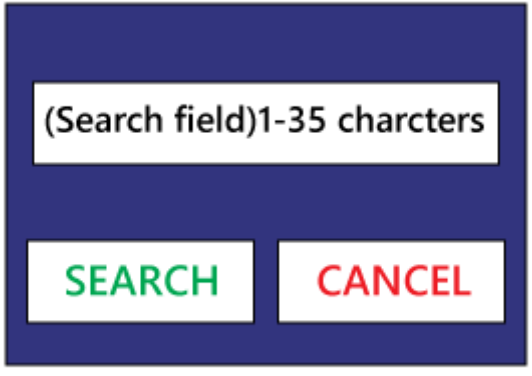
- This is a very focused approach where ***only the modified section*** goes under the regression test instead of the impact region

In the below application, and in the first build, the developer develops the Search button that accepts 1-15 characters. Then the test engineer tests the Search button with the help of the test case design technique.



Build 1-B001

Now, the client does some modification in the requirement and also requests that the Search button can accept the 1-35 characters. The test engineer will test only the Search button to verify that it takes 1-35 characters and does not check any further feature of the first build.

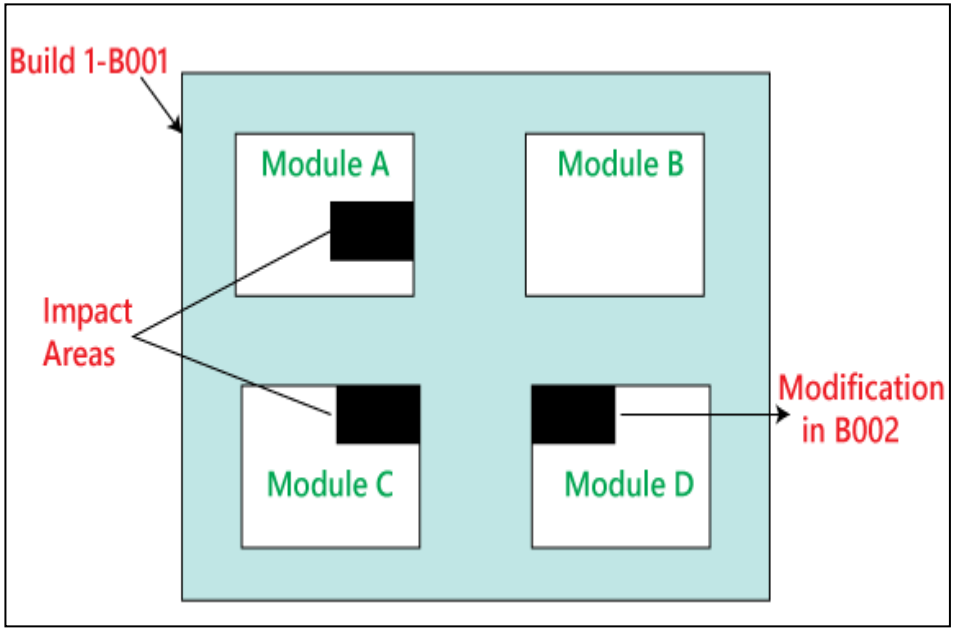


Build 1-B002

2) Regional Regression Testing [RRT]

- In regional regression testing, the ***modification and impact areas are tested***.
- This area is examined to find out if any dependable modules might be affected by the changes.

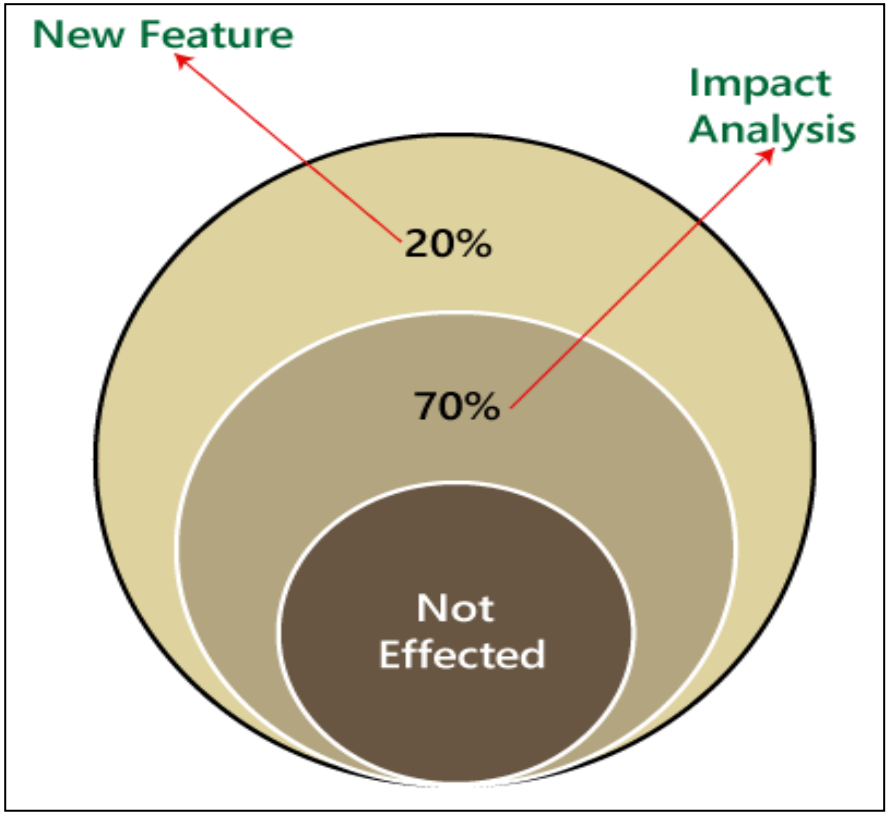
In the below image as we can see that we have four different modules, such as Module A, Module B, Module C, and Module D, which are provided by the developers for the testing during the first build. Now, the test engineer will identify the bugs in Module D. The bug report is sent to the developers, and the development team fixes those defects and sends the second build.



In the second build, the previous defects are fixed. Now the test engineer understands that the bug fixing in Module D has impacted some features in Module A and Module C. Hence, the test engineer first tests the Module D where the bug has been fixed and then checks the impact areas in Module A and Module C.

3) Full Regression Testing [FRT]

- Full Regression Testing (FRT) checks all the features of an application, both new and old.
- It is usually done in later releases and before launching.
- FRT ensures that modified features work correctly without breaking existing ones.



Test Plan

- A Test Plan is a **comprehensive document** outlining the ***policies***, ***goals***, ***timeline***, ***equipment***, ***technology***, ***estimates***, ***due dates***, and ***manpower*** that will be used to perform testing for the software products.
- A test plan is a document that ***consists of all future testing-related activities***.
- In any company whenever a new project is taken up before the tester is involved in the testing the test manager of the team would prepare a test Plan.
- The test plan serves as the ***blueprint that changes according to the progressions in the project and stays current at all times***.
- It serves as a base for conducting testing activities and coordinating activities among a QA team.

Types of Test Plan in Software Testing

- There are three types of test plans

Master Test Plan

- This includes the ***overall plan***, ***roadmap***, and software testing life cycle ***outline***.

Phase Test Plan

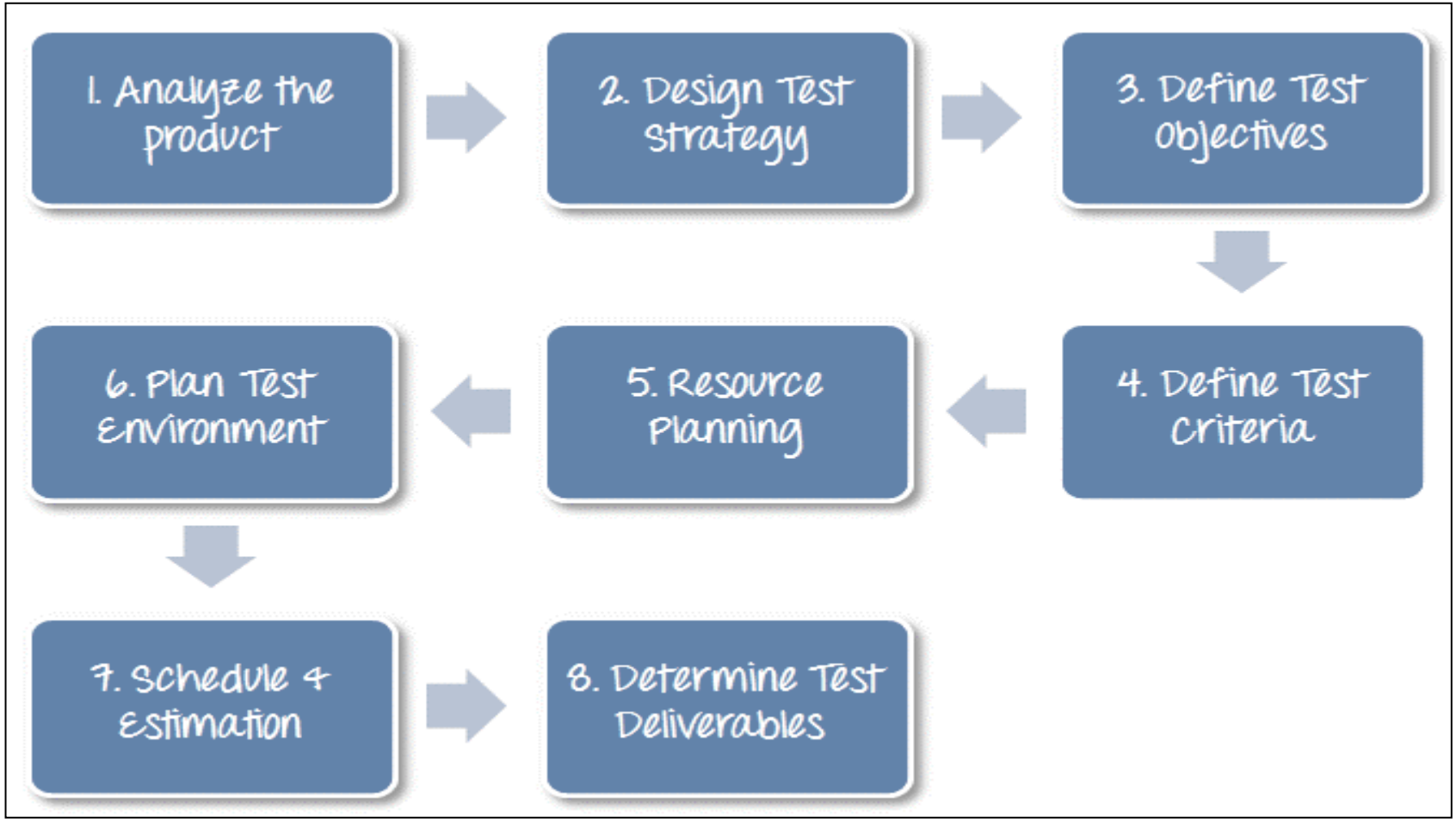
- As a test plan gets ***converted to a sprint***.
- Here the focus is on one sprint at a time.

Specific Test Plan

- This test plan is around a ***particular test case scenario*** or tool.
- It has details of that one functionality and how it should be tested.

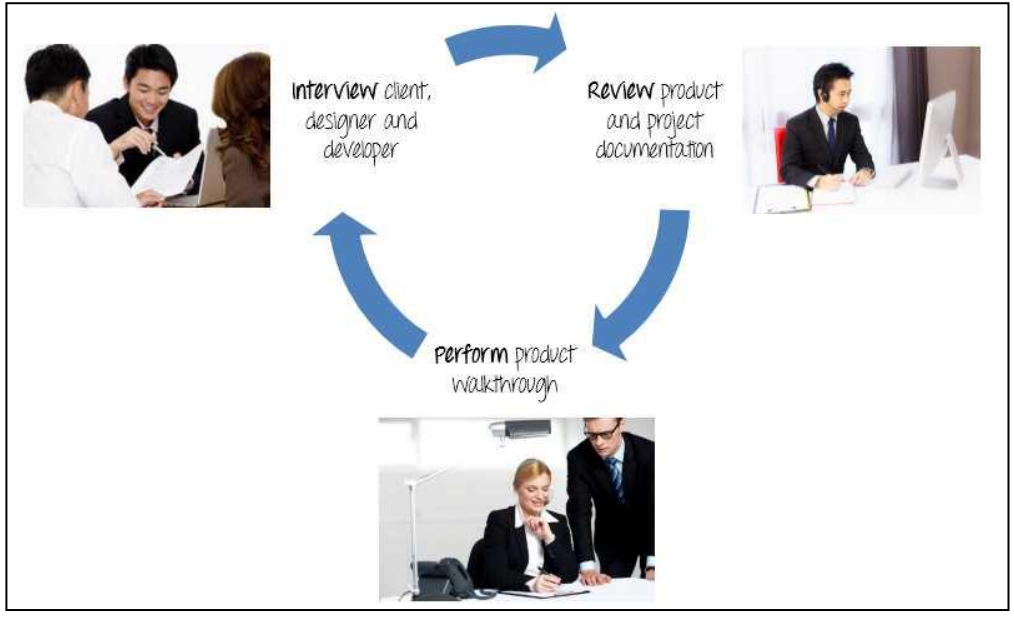
How to write a Test Plan

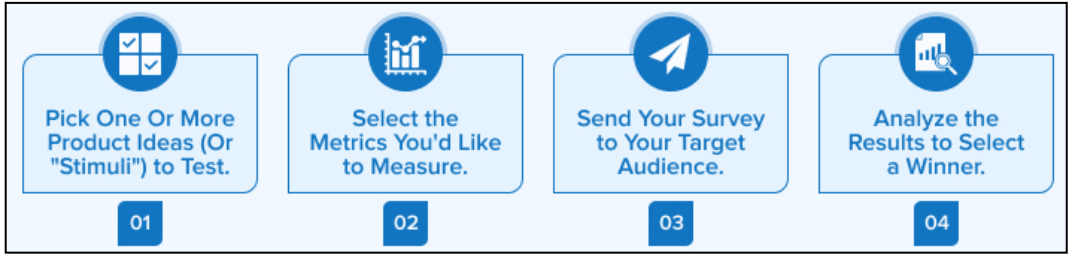
- According to IEEE 829, follow the following steps to prepare a test plan.



Step 1) Analyze the product

- This phase focuses on ***analyzing the product***, ***Interviewing clients***, ***designers***, and ***developers***, and performing a product walkthrough.
- This stage focuses on answering the following questions:
  - ✓ ***What is the primary objective of the product?***
  - ✓ ***Who will use the product?***
  - ✓ ***What are the hardware and software specifications of the product?***
  - ✓ ***How does the product work?***

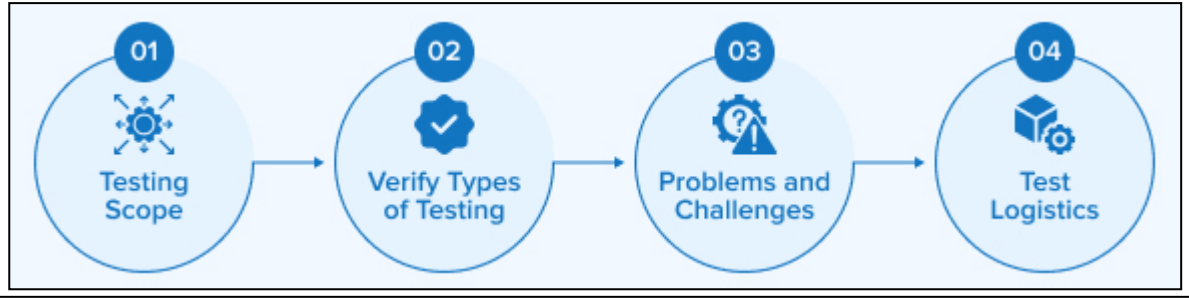


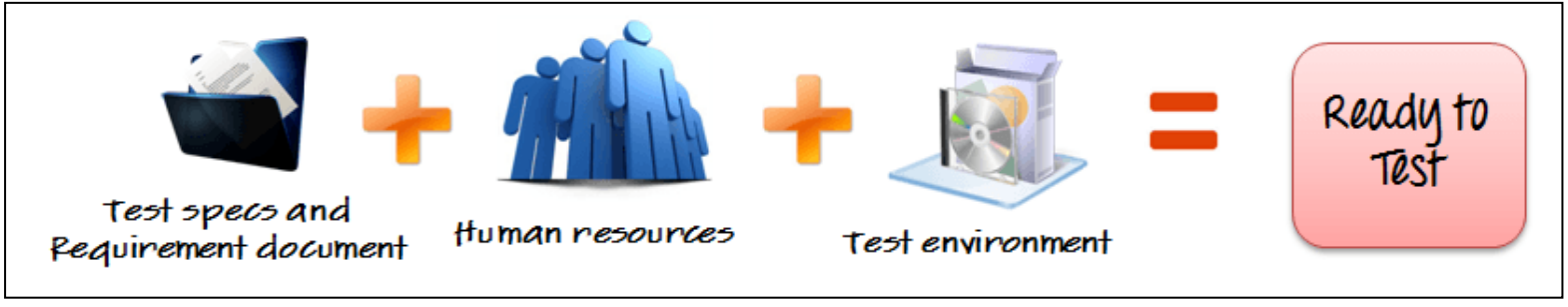


Step 2) Develop Test Strategy

- The test strategy document is **prepared by the manager** and details the following information:
  - ***Scope of testing*** which means the components that will be tested and the ones that will be skipped.
  - ***Type of testing*** which means different types of tests that will be used in the project.
  - ***Risks and issues*** that will list all the possible risks that may occur during testing.
  - ***Test logistics*** mentions the names of the testers and the tests that will be run by them.

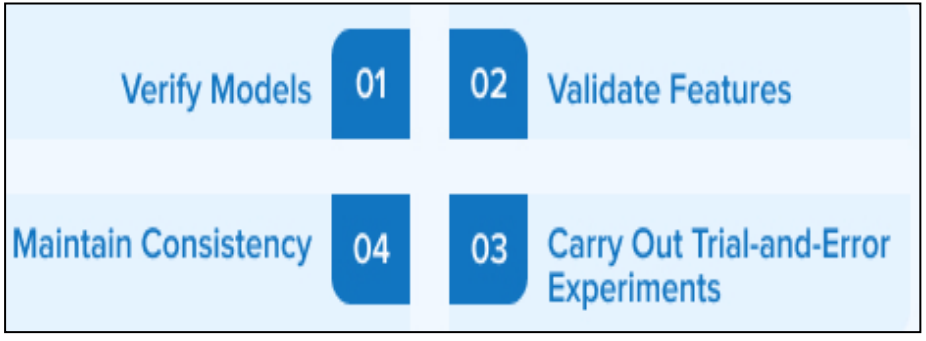
You will start to test when you have all required items



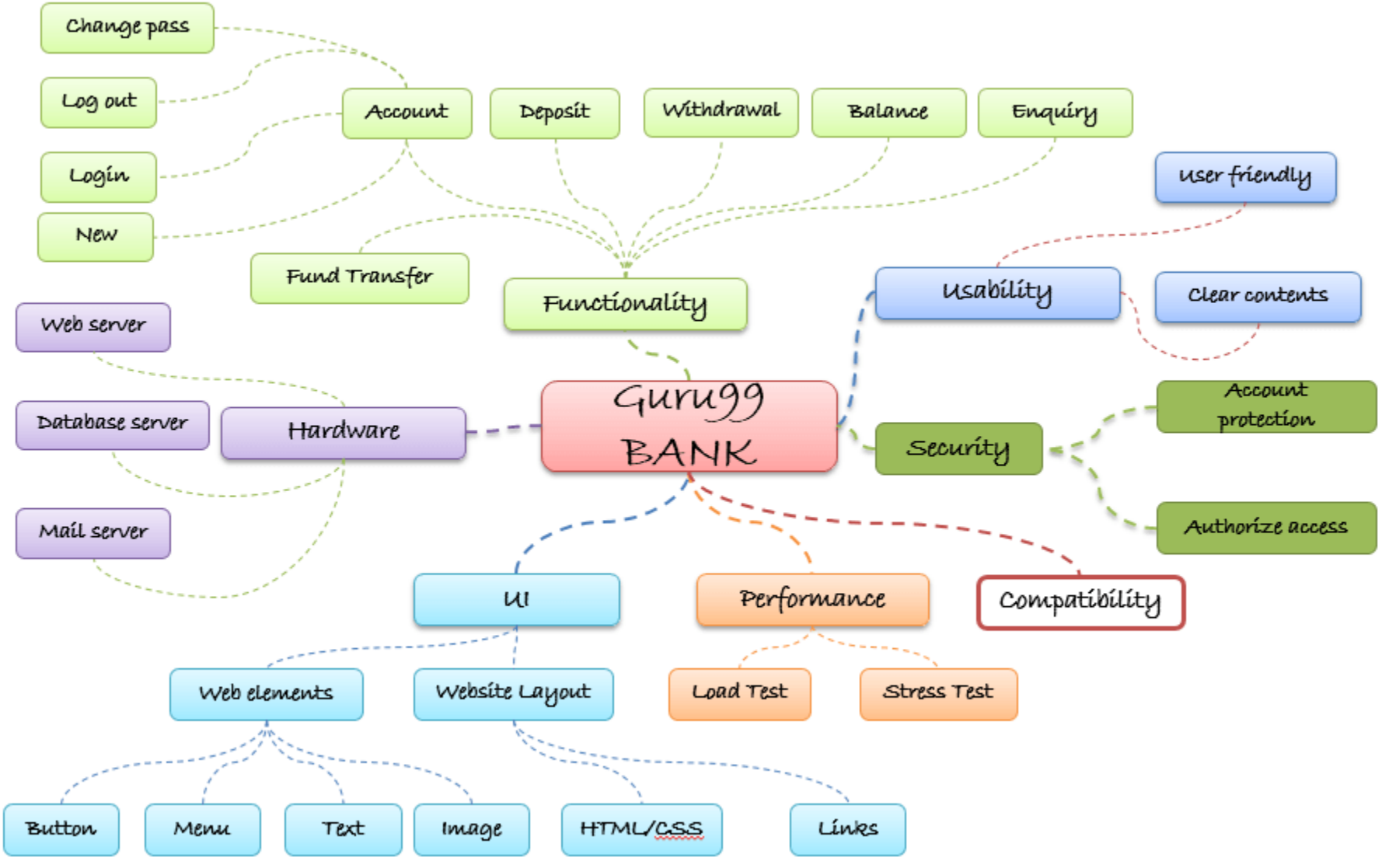


Step 3) Define Test Objective

- This phase ***defines the objectives*** and ***expected results*** of the test execution.
- Objectives include:
  - ✓ A **list of software features** like functionality, GUI, performance standards, etc.
  - ✓ The **ideal expected outcome** for every aspect of the software that needs testing.



Functionality (core operations, integrations, workflows)
User Interface (GUI elements, navigation, responsiveness)
Performance (speed, load handling, resource usage)
Security (data protection, authentication, vulnerability testing)
Compatibility (cross-platform, device adaptability)
Usability (user-friendliness, accessibility, ease of use)


Change pass
Log out
Login
New
Account
Deposit
Withdrawal
Balance
Enquiry
Fund Transfer
Functionality
User friendly
Usability
Clear contents
Web server
Database server
Hardware
Mail server
Guru99 BANK
Security
Account protection
Authorize access
UI
Performance
Compatibility
Web elements
Website Layout
Load Test
Stress Test
Button
Menu
Text
Image
HTML/CSS
Links

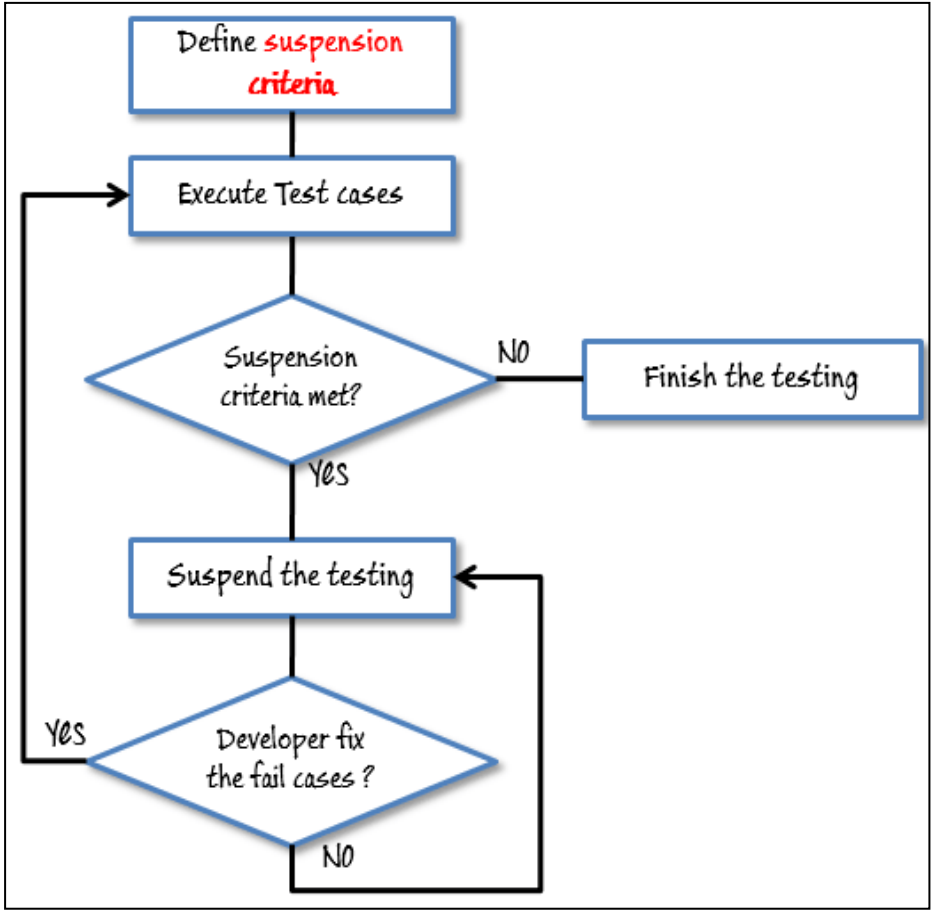
Step 4) Define Test Criteria

- Test Criteria is a ***standard or rule*** on which a test procedure or test judgment can be based.

- There're 2 types of test criteria as following

  (i) Suspension Criteria

  - Suspension criteria ***define the benchmarks*** for suspending all the tests.
  - Example: If your team members report that there are 40% of test cases failed, you should suspend testing until the development team fixes all the failed cases.



(ii) Exit criteria:

- Exit criteria define the benchmarks that signify the successful completion of the test phase or project.
- These are expected results and must match before moving to the next stage of development.

Step 5) Resource Planning

- This phase aims to ***create a detailed list of all the resources required*** for project completion.
- Resource could be ***human***, ***equipment*** and ***materials*** needed to complete a project

Human Resource

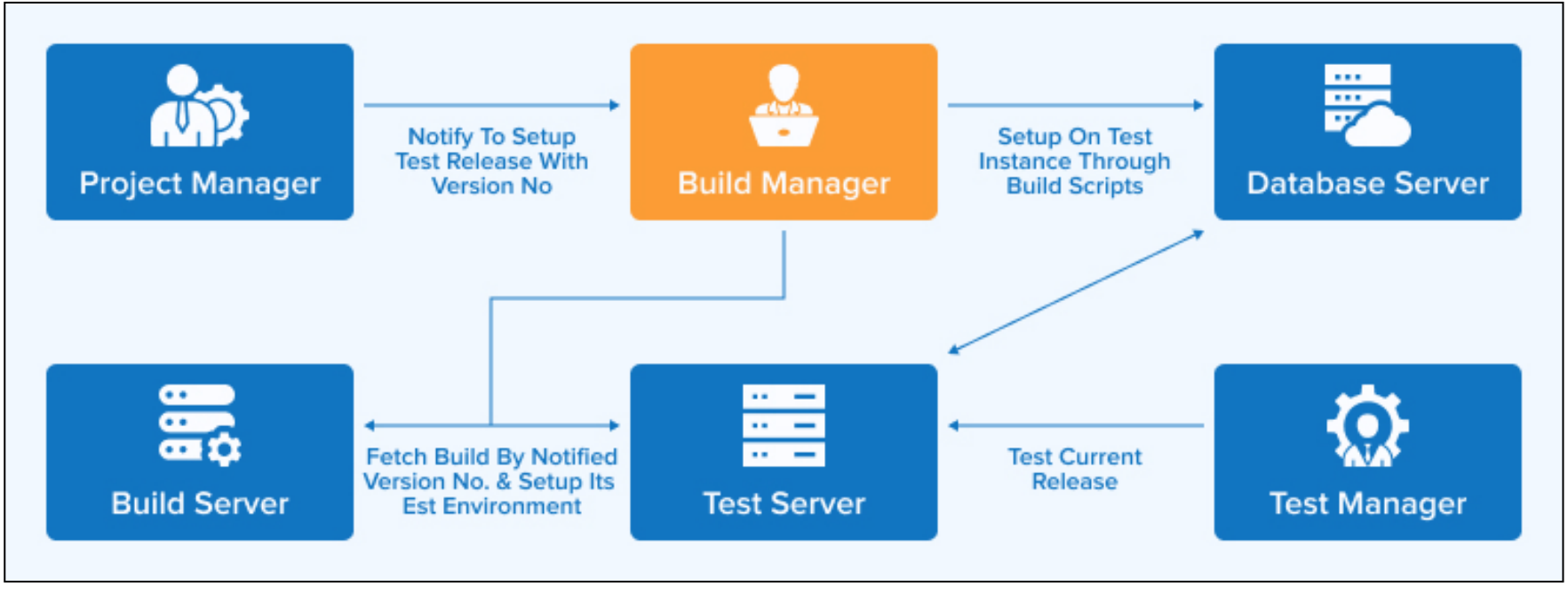
No.	Member
1	Test Manager
2	Tester
3	Test Administrator
4	SQA members

System Resource

No.	Member
1	Server
2	Test tool
3	Network
4	Computer

Step 6) Plan Test Environment

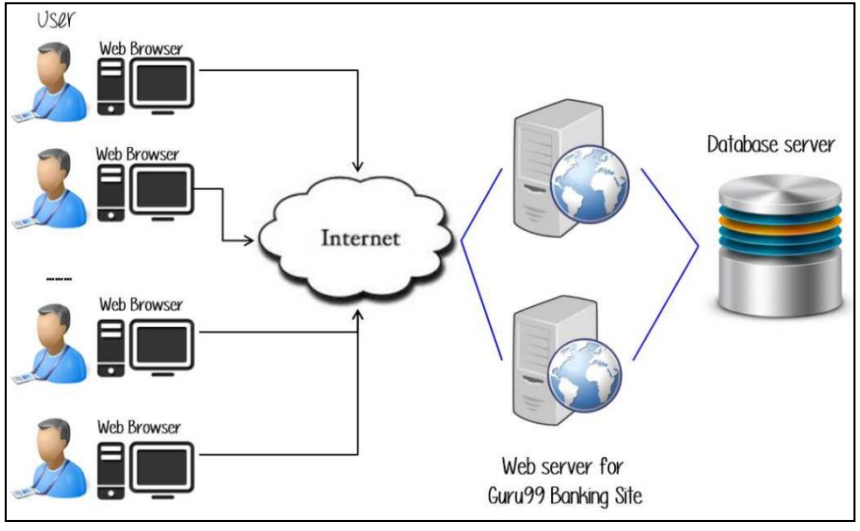
- A testing environment is a ***setup of software and hardware*** on which the testing team is going to execute test cases.
- The test environments must be real devices, installed with real browsers and operating systems so that testers can monitor software behavior in real user conditions.



- To finish this task, you need ***a strong cooperation*** between **Test Team** and **Development Team**

You should ask the developer some questions to understand the web application under test clearly. Here're some recommended questions. Of course, you can ask the other questions if you need.

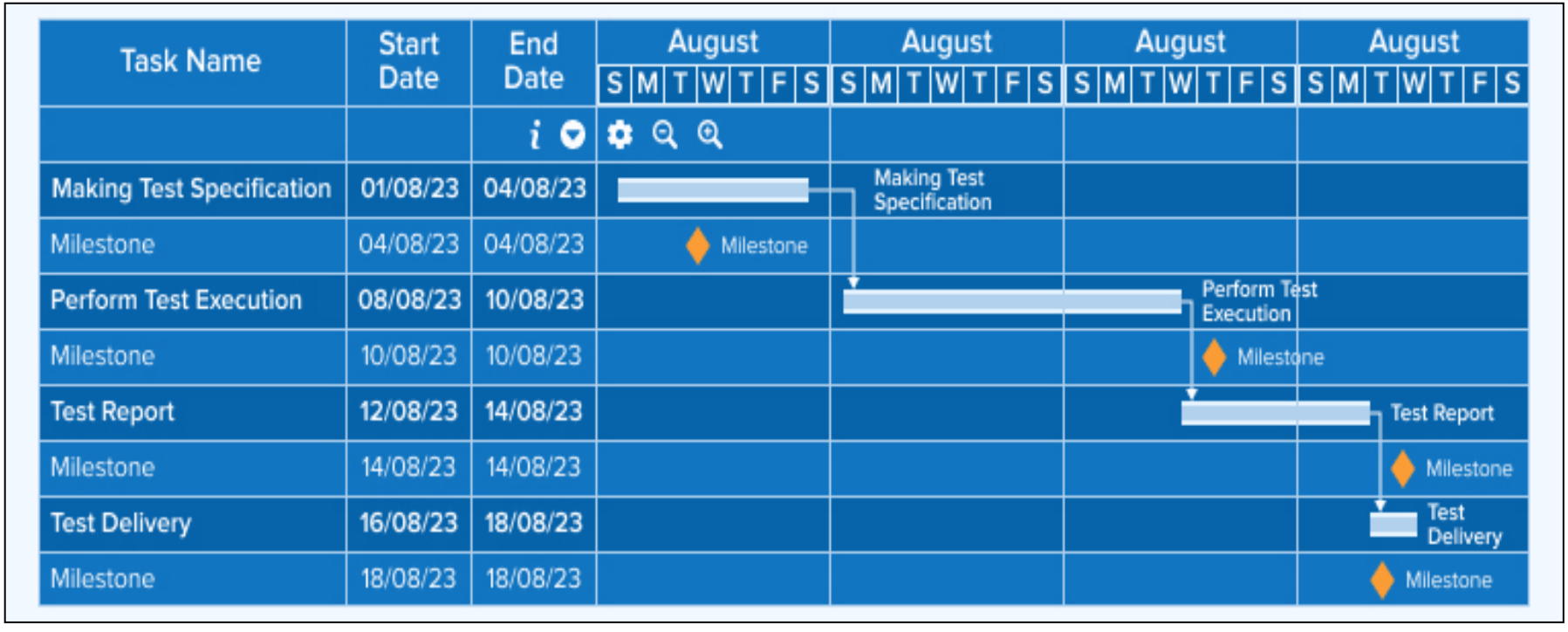
What is the maximum user connection which this website can handle at the same time?
What are hardware/software requirements to install this website?
Does the user's computer need any particular setting to browse the website?



Step 7) Schedule & Estimation

- ***Break down the project into smaller tasks and allocate time and effort for each task.***
- This helps in efficient time estimation.
- Create a schedule to complete these tasks in the designated time with a specific amount of effort.

Task	Members	Estimate effort
Create the test specification	Test Designer	170 man-hour
Perform Test Execution	Tester, Test Administrator	80 man-hour
Test Report	Tester	10 man-hour
Test Delivery		20 man-hour
Total		**280 man-hour**


Task Name
Start Date
End Date
August
S M T W T F S
Making Test Specification 01/08/23 04/08/23
Milestone 04/08/23 04/08/23
Perform Test Execution 08/08/23 10/08/23
Milestone 10/08/23 10/08/23
Test Report 12/08/23 14/08/23
Milestone 14/08/23 14/08/23
Test Delivery 16/08/23 18/08/23
Milestone 18/08/23 18/08/23
Making Test Specification
Milestone
Perform Test Execution
Milestone
Test Report
Milestone
Test Delivery
Milestone

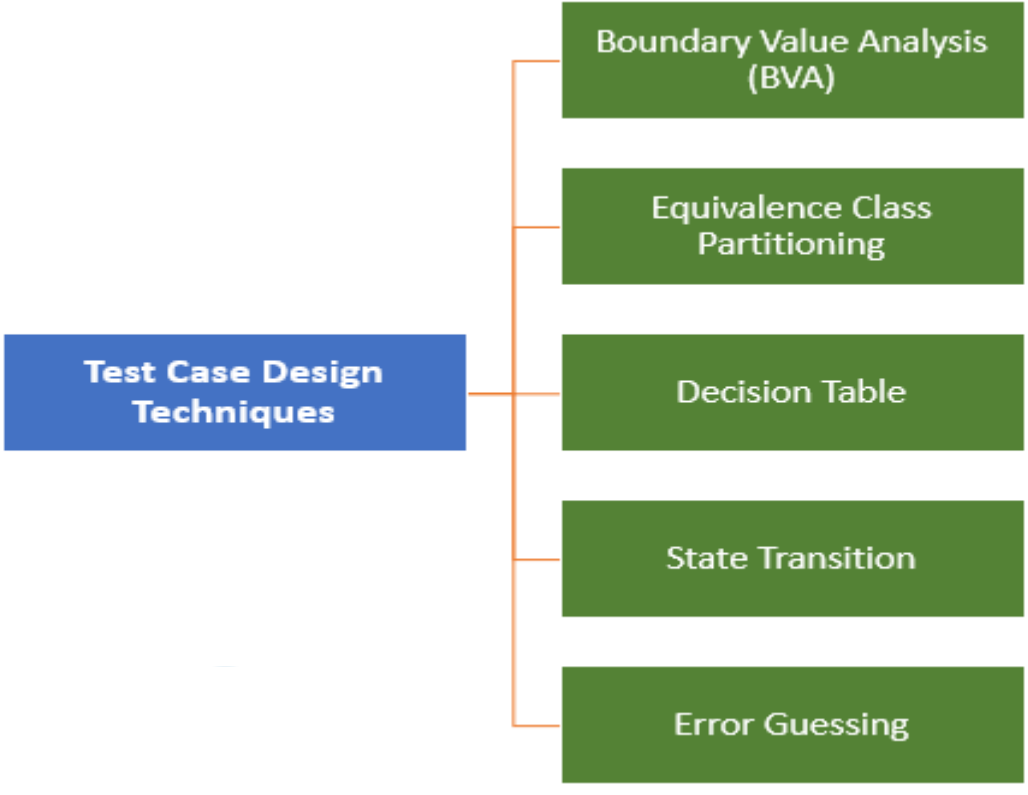
Step 8) Test Deliverables

- Test deliverables refer to the ***list of documents***, ***tools***, and ***other equipment*** that must be created, provided, and maintained to support testing activities in the project.

Deliverables required before testing	Deliverables required during testing	Deliverables required after testing
Test Plan	Test Scripts	Test Results
Test Design	Simulators	Defect Reports
	Test Data	Release Notes
	Error and Execution Logs	

Test Design

- Test design is a process that ***defines how testing has to be done***.
- It involves the process of identifying the
  - ***Testing Techniques***
  - ***Test scenarios***
  - ***Test cases***
  - ***Test data***
  - ***Expected test results.***



Test Design Techniques

Steps in Test Design Phase

1. Analyze Requirements

– Review the software requirements to identify what needs to be tested.

2. Identify Test Scenarios

– Based on the requirements, create a list of test scenarios to be covered during testing.

3.Design Test Cases

– Develop test cases for each scenario, detailing the input data, expected output, and test procedure.

4. Create Test Data

– Prepare the required test data to be used in executing the test cases.

5. Map Test Cases to Requirements

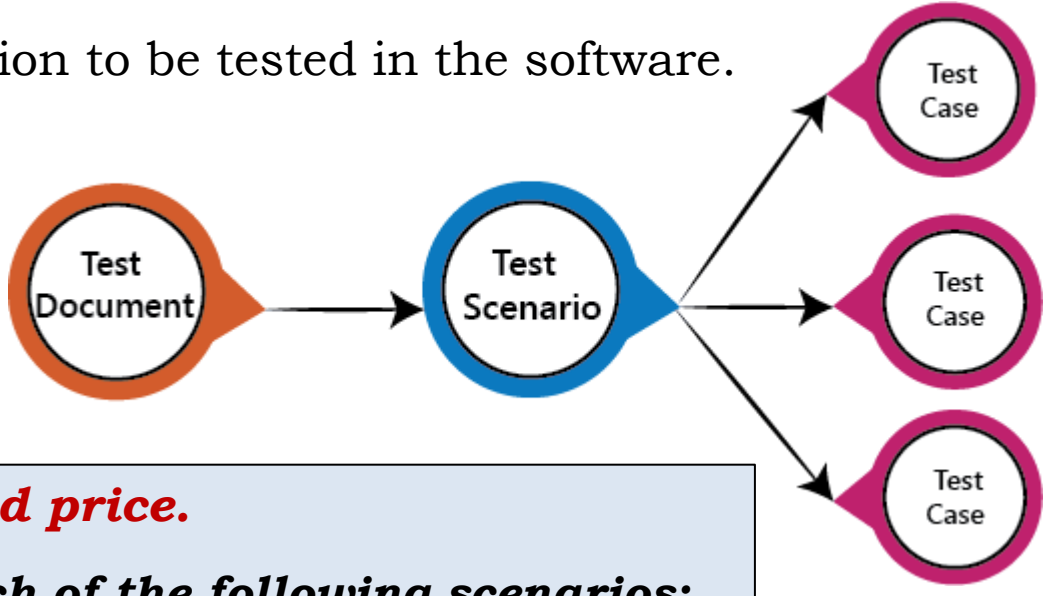
– Trace test cases back to their corresponding requirements to ensure complete coverage.

Test Case

- A test case is a document which has a set of conditions or actions that are performed on the software application in order to verify the expected functionality of the feature.

Test Scenarios

- A scenario is a specific condition to be tested in the software.



Let's say you have a field called price.

You will write a test case for each of the following scenarios:

- ***Users should not be able to enter non-numeric characters.***
- ***Price should be greater than zero.***
- ***Price should be less than a million dollars.***

Test case template

Header

Test Case Name/ID :-Release - Version - Application Name - Module

Test Case Type:- F.T.C | I.T.C | S.T.C

Requirement Number:-

Module:-

Serverity:- Critical/Major/Minor

Status:-

Release:-

Version:-

Pre-condition:-

Test Data:-

Summary:-

Body

Step No.	Descri-pation	Inputs	Expected Result	Actual Reasult	Status	Comments
...	...	...	...	...	...	...
...	...	...	...	...	...	...

Footer

Author:- Reviewd By:-

Date:- Approved By:-

Fields	Description
Test Case ID	Each test case should have a unique ID.
Test Case Description	Each test case should have a proper description to let testers know what the test case is about.
Pre-Conditions	Conditions that are required to be satisfied before executing the test case.
Test Steps	Mention all test steps in detail and to be executed from the end-user's perspective.
Test Data	Test data could be used as input for the test cases.
Expected Result	The result is expected after executing the test cases.
Post Condition	Conditions need to be fulfilled when the test cases are successfully executed.
Actual Result	The result that which system shows once the test case is executed.
Status	Set the status as Pass or Fail on the expected result against the actual result.

Fields	Description
Project Name	Name of the project to which the test case belongs.
Module Name	Name of the module to which the test case belongs.
Reference Document	Mention the path of the reference document.
Created By	Name of the tester who created the test cases.
Date of Creation	Date of creation of test cases.
Reviewed By	Name of the tester who reviews the test case.
Date of Review	When the test cases were reviewed.
Executed By	Name of the tester who executed the test case.
Date of Execution	Date when the test cases were executed.
Comments	Include comments which help the team to understand the test cases.

	A	B	C	D	E	F	G
1	**Test case tamplate**						
2	**test case name**	Delta-3.0-ICICI-Login					
3	**test case type**	Functional test case					
4	**requirement no**	1					
5	**module**	login					
6	**status**	XXX					
7	**severity**	critical					
8	**release**	Delta					
9	**version**	3					
10	**pre-condition**	required one login					
11	**test data**	username-abc, password-123					
12	**summary**	to check the functionality of login					
13	**Steps no**	**Description**	**Inputs**	**Expected result**	**Actual results**	**Status**	**Comments**
14	**1**	open "**Browser**" and enter the "**Url**"	https://QA/Main//	"Login page must be display	As Expected	pass	XXX
15	**2**	enter the following values for "**Username**" :					
16		Valid(abc)	abc	**Accpect**	Login page must be disp	pass	XXX
17		Invalid	555	Error message "**invalid login**"	not as expected	fail	bug #1
18		**Blank**	Null	Error message username cannot b	not as expected	fail	
19		**Symbols**	2 alphabet	Error message invalid login	not as expected	fail	
20							
21	**3**	enter the following values for "**Password**":					
22		valid	123	**Accpect**	Login page must be disp	pass	XXX
23		invalid	xy3	Error message invalid login	not as expected	fail	
24		Blank		Error message password cannot b	not as expected	fail	
25	**4**	enter the valid username and password and click on "**OK**" button	abc,123	"**home Pag** " must be displayed	**home page** is displayin	pass	
26	**5**	enter the valid username and password and click on "**Cancel**" button	abc,123	all field must be cleard	the entered data is clea	pass	XXX
27							
28	**author**	test engineer name					
29	**date**	1/4/2020					
30	**reviewed by**	ryan					
31	**apporved by**	jessica					
32							
33							

Project Name	Bank Webiste Functionality								
Module Name	Login Functionality								
Created By	Archana								
Created Date	yyyy-mm-xx								
Executed By	XYZ								
Executed Date	YYYY-MM-DD								
Test Case ID	Test Case Description	Pre Steps	Test Step	Preconditions	Test Data	Expected Result	Actual Result	Status	Comments
Test the Login Functionality in Banking	Verify login functionality with valid username & password		Navigate to login page			Able to see the login page	As expected	Pass	
			Enter valid username	Valid Username	username: choudaryac97@gmail.com	Credential can be entered	As expected	Pass	
			Enter valid password	Valid password	password: XXXXXXX@1	Credential can be entered	As expected	Pass	
			Click on login button			User logged	User logged successfully	Pass	
Test the Login Functionality in Banking	Verify login functionality with valid username & invalid password		Navigate to login page			Able to see the login page	As expected	Pass	
			Enter valid username	Valid Username	username: choudaryac97@gmail.com	Credential can be entered	As expected	Pass	
			Enter valid password	Invalid password	password: XXXXXXX@2	Credential can be entered	As expected	Pass	
			Click on login button			User logged	Unsuccessful login	Fail	

Test Case Type	Description	Test Step	Expected Result	Status
Functionality	Area should accommodate up to 20 characters	Input up to 20 characters	All 20 characters in the request should be appropriate	Pass or Fail
Security	Verify password rules are working	Create a new password in accordance with rules	The user's password will be accepted if it adheres to the rules	Pass or Fail
Usability	Ensure all links are working properly	Have users click on various links on the page	Links will take users to another web page according to the on-page URL	Pass or Fail

Example : Test case to check the login functionality

Test Case ID	Test Case Description	Test Steps	Test Data	Expected Result	Actual Result	Status
TU01	Check user login when email ID and password are entered	1. Go to website 2. *https://id.testsigma.com/ui/login* 3. Enter email ID 4. Enter password 5. Click submit	Email – sample@gmail.com Password – Sample@123	User should be able to login	Login was successful	Pass
TU02	Check user login when email ID and password are entered	1. Go to website 2. *https://id.testsigma.com/ui/login* 3. Enter email ID 4. Enter password 5. Click submit	Email – sample2@gmail.com Password – Sample@321	User should not be able to login	Login was not successful	Pass

Unit Test Case

Checking if the username validates at least for the length of eight characters.

Test Id	Test Condition	Test Steps	Test Input	Test Expected Result	Actual Result	Status
1.	Check if the username field accepts the input of thirteen characters.	Give input	abcdefg hijklm	Accepts for thirteen characters.	Accepts for thirteen characters	Pass

Functionality Test Case

Checked whether the username and password both work together on the login click.

Test Id	Test Condition	Test Steps	Test Input	Test Expected Result	Actual Result	Status
1.	Check that with the correct username and password able to log in.	1. Enter the username 2. Enter the password 3. Click the login	username: abcdefghijklm password: welcome	Login successful	Login successful	Pass

User Acceptance Test Case

The user feedback is taken if the login page is loading properly or not.

Test Id	Test Condition	Test Steps	Test Input	Test Expected Result	Actual Result	Status	Remarks
1.	Check if the loading page loading efficiently for the client.	1. Click on the login button.	None	Welcome to the login page.	Welcome to the login page.	Fail	The login page is not loaded due to a browser compatibility issue on the user's side.

Test Cases for VTOP Login Page

Module : Login
Pre-conditions :User must have a registered VTOP account

Test Id	Test Condition	Test Steps	Test Input	Test Expected Result	Actual Result	Status
TC_001	Verify login with valid credentials	1. Open the login page 2. Enter valid username and password 3. Click "Login"	Username: user123 Password: Pass@123	User should be redirected to the dashboard	User successfully logged in and redirected to the dashboard	Pass
TC_002	Verify login with invalid credentials	1. Open the login page 2. Enter invalid username and password 3. Click "Login"	Username: user123 Password: WrongPass	Error message "Invalid username or password" should be displayed	No error message was displayed	Fail
TC_003	Verify password masking	1. Open the login page 2. Enter password in the field	Password: Pass@123	Password should be masked with dots or asterisks	Password was displayed in plain text	Fail
TC_004	Verify forgot password functionality	1. Click on "Forgot Password" 2. Enter registered email 3. Click "Submit"	Email: user@example.com	Password reset link should be sent to the registered email	Reset email received successfully	Pass

Module : Security

Pre-conditions :None, User must have a registered email/phone number

Test Id	Test Condition	Test Steps	Test Input	Test Expected Result	Actual Result	Status
TC_005	Verify password masking in input field	1. Open VTOP login page 2. Enter password in the password field	Password: MySecretPass123	Password should be masked with dots or asterisks	Password is masked correctly with dots (••••••••••••••)	Pass
TC_006	Verify "Forgot Password" functionality	1. Open VTOP login page 2. Click "Forgot Password" 3. Enter registered email/phone 4. Submit request	Email: user@example.com OR Phone: +1234567890	Password reset link or OTP is sent successfully	Password reset link received successfully	Pass
TC_007	Verify CAPTCHA functionality (if present)	1. Open VTOP login page 2. Enter valid credentials 3. Enter incorrect CAPTCHA 4. Click "Login"	Username: valid_user Password: valid_pass CAPTCHA: incorrect_value	Login should fail with an error message	Login failed with an error message "Incorrect CAPTCHA, please try again"	Pass

Module : Session Management
Pre-conditions :User must be logged in

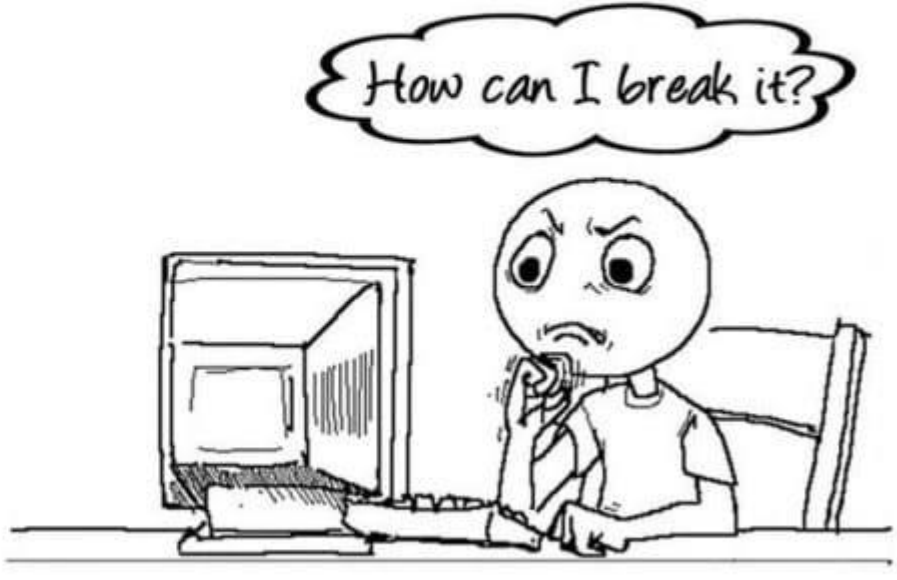
Test Id	Test Condition	Test Steps	Test Input	Test Expected Result	Actual Result	Status
TC_008	Verify session timeout after inactivity	1. Log in to VTOP 2. Stay inactive for a specific period (e.g., 10 minutes)	No user activity for 10 minutes	User is logged out automatically, and the login page is displayed	User was logged out and redirected to the login page	Pass

Mistakes to Avoid While Writing Test Cases

- Making test cases too specific
- Limiting test cases according to user roles.
- Not categorizing test cases.
- Poor test cases are too dependent on the internal working of the software and hence will break if there is an internal change.
- Confusing between unit test and integration test.
- Your test cases do not consider end users.
- Need to write more test cases to cover most areas of the system.
- Last, not maintaining the test cases as the software changes.

DEVELOPER

TESTER



Test Execution

- Test Execution is the ***process of running test cases*** to verify if the developed code, functions, or modules work as expected based on client or business requirements.
- It takes place after development, during the testing phase, where different testing techniques are applied, and test cases are created and executed.
- The tester will usually write or execute a certain number of test cases, and test scripts or do automated testing.
  - If it ***creates any errors then it will be informed to the respective development team*** to correct the issues in the code.
  - If the ***text execution process shows successful results then it will be ready for the deployment phase*** after the proper setup for the deployment environment.

1. Test Execution Guidelines

- The following are important aspects of test execution that need to be understood before running test cases:

(i) Build

- Build is ***standalone software***.
- A ***source code is converted into a build*** that is deployed on the test environment.
- This build is tested for bugs and quality assurance.

(ii) Test Environment

- There is a ***dedicated server*** for testing purposes.
- After the application is tested, it gets deployed to another server for production usage.
- **Saucelabs** and **Browserstack** provide these test environments as a service model.

(iii) Test Team Size

- Test team size is ***dynamic***.
- The initial project is on test lead, then QA and senior QA are assigned tasks by the lead.
- The team size keeps changing based on company projects and deadlines.

(iv) Test Cycles:

- The testing process also happens in a cycle.
- **Initial phase**, the product is ***tested to ensure that functionalities*** are working and no other bugs are left.
- **Next cycle**, the ***system is retested for experimental testing*** and finding deeper bugs.

(v) Test Execution Phase:

- Test execution consists of ***test scripts***, ***maintenance*** and ***bug reporting***.
- After testing, the environment is configured, and the build is deployed.

(vi) Smoke and Sanity test cases:

- When testing is done on a build, the testing process comes under smoke and sanity testing.
- **Sanity testing** is done to ***ensure that functionalities are working in the build***.
- **Smoke testing** ***ensures that all the bugs have been fixed*** in the build deployed.

(vii) Defect Reporting:

- The bug life cycle is an important aspect of test execution.
- Bugs found are reported and maintained in a separate document.
- **Bugzilla** and **JIRA** are used for bug report management.

2. Test Execution States

- Some important test execution states/results have a definite meaning to them.

States	Description
Pass	Test case is executed, and the expected result is the same
Fail	Test case is executed, and the expected result is not the same
Inconclusive	Test case is executed, and there is no clear result
Block	Test case cannot be executed because one of the test case preconditions is not met.
Deferred	Test case is not executed yet and will run in the future.
In progress	Test case is currently running.
Not run	Test case has not been executed yet.

3. Activities for Test Execution

- The following are the 5 main activities that should be carried out during the test execution.

(i) Defect Finding and Reporting

- Defect finding is the process of ***identifying bugs*** or ***errors*** while testing the code.
- If a test case fails or an error appears, it is recorded and reported to the development team.
- **End users** may also find and report errors during user acceptance testing.
- The respective team will review the recorded errors and work on fixing them.

(ii) Defect Mapping

- After an error is detected and reported, the development team fixes it as needed.
- Then, the testing team runs test cases again on the updated code to ensure it works correctly.

(iii) Re-Testing

- Re-testing ensures a smooth release by ***testing modules*** or the ***entire product*** again.
- If a **new feature** is added after release, all modules are re-tested to prevent new defects.

(iv) Regression Testing

- Regression Testing checks if recent code changes work correctly.
- It ensures that new modules or functions do not affect the normal operation of the application or product.

(v) System Integration Testing:

- System Integration Testing checks whether all components or modules of a system work together as a whole.
- Instead of testing each part separately, it ensures everything functions correctly in a single test environment.

4. Test Execution Process

- The test execution technique has three phases that help process the test results and confirm their accuracy.

1. Creation of Test Cases

- The first phase is to ***create suitable test cases for each module or function***.
- Tester with good domain knowledge is essential to create suitable test cases.
- Test cases should be simple and created on time to avoid delays in product release
- The created test cases should not be repeated again.
- It should cover all the possible scenarios raised in the application.

2. Test Cases Execution

- After test cases have been created, execution of test cases will take place.
- **Quality Analyst team** will do automated or manual testing depending upon the test case scenario.
- It is always preferable to do both automated as well as manual testing to have 100% assurance of correctness.
- The ***selection of testing tools*** is also important to execute the test cases.

3. Validating Test Results

- Execute the test cases and record the results in a separate file or report.
- Compare the actual results with the expected results.
- Note down the time taken to complete each test case.
- If any test case fails or does not meet the expected outcome, report it to the development team for validation.

5. Test Execution Report

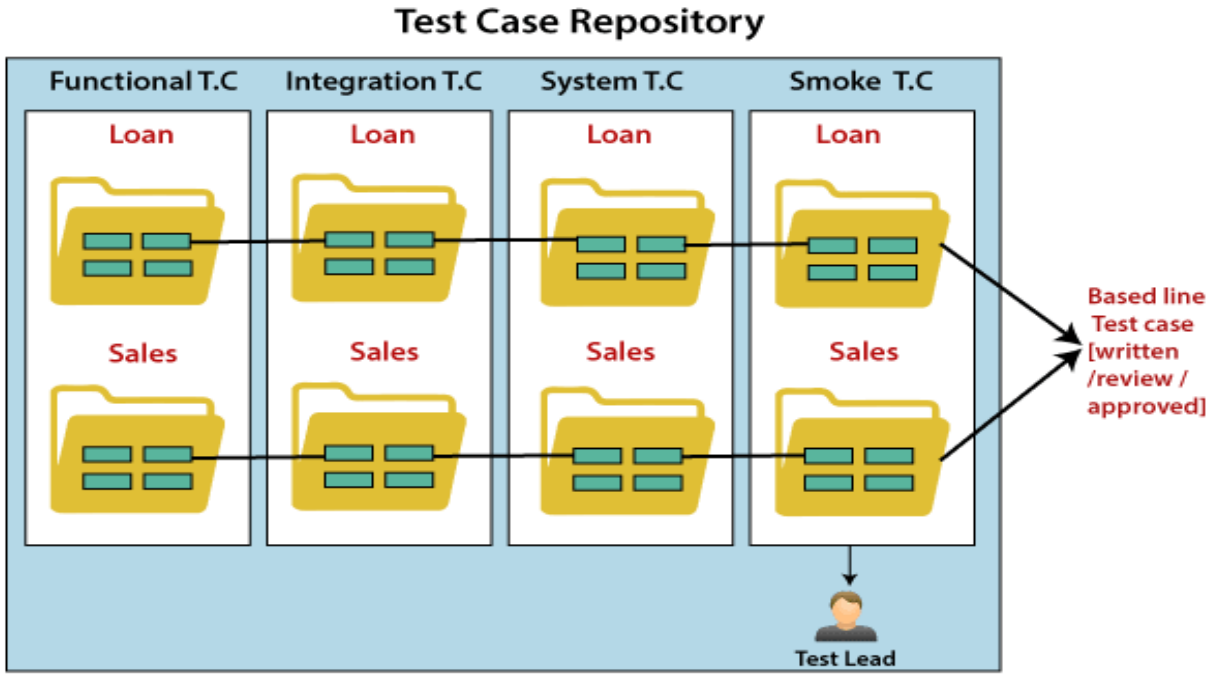
- The Test Execution Report is nothing but a document that contains all the information about the test execution process. It is documentation that will be recorded and updated by the QA team. The documentation or the report contains various information. They are:
  - ***Who all are going to execute the test cases?***
  - ***Who is doing the unit testing, integration testing, system testing, etc.,***
  - ***Who is going to write test cases?***
  - ***The number of test cases executed successfully.***
  - ***The number of test cases failed during the testing.***
  - ***The number of test cases executed today.***
  - ***The number of test cases yet to be executed.***
  - ***What are the automation test tools used for today's test execution?***
  - ***What are the modules/functions testing today?***
  - ***Recording the issues while executing the test cases.***
  - ***What is today's testing plan?***
  - ***What is tomorrow's testing plan?***
  - ***Recording the pending plans.***
  - ***Overall success rate.***
  - ***Overall failure rate.***

Test Review

- A Test Review is a formal process to check software and ensure that recent changes work correctly.
- The reviewer verifies the **correctness**, **accuracy**, **flow**, and **coverage** of the test case.

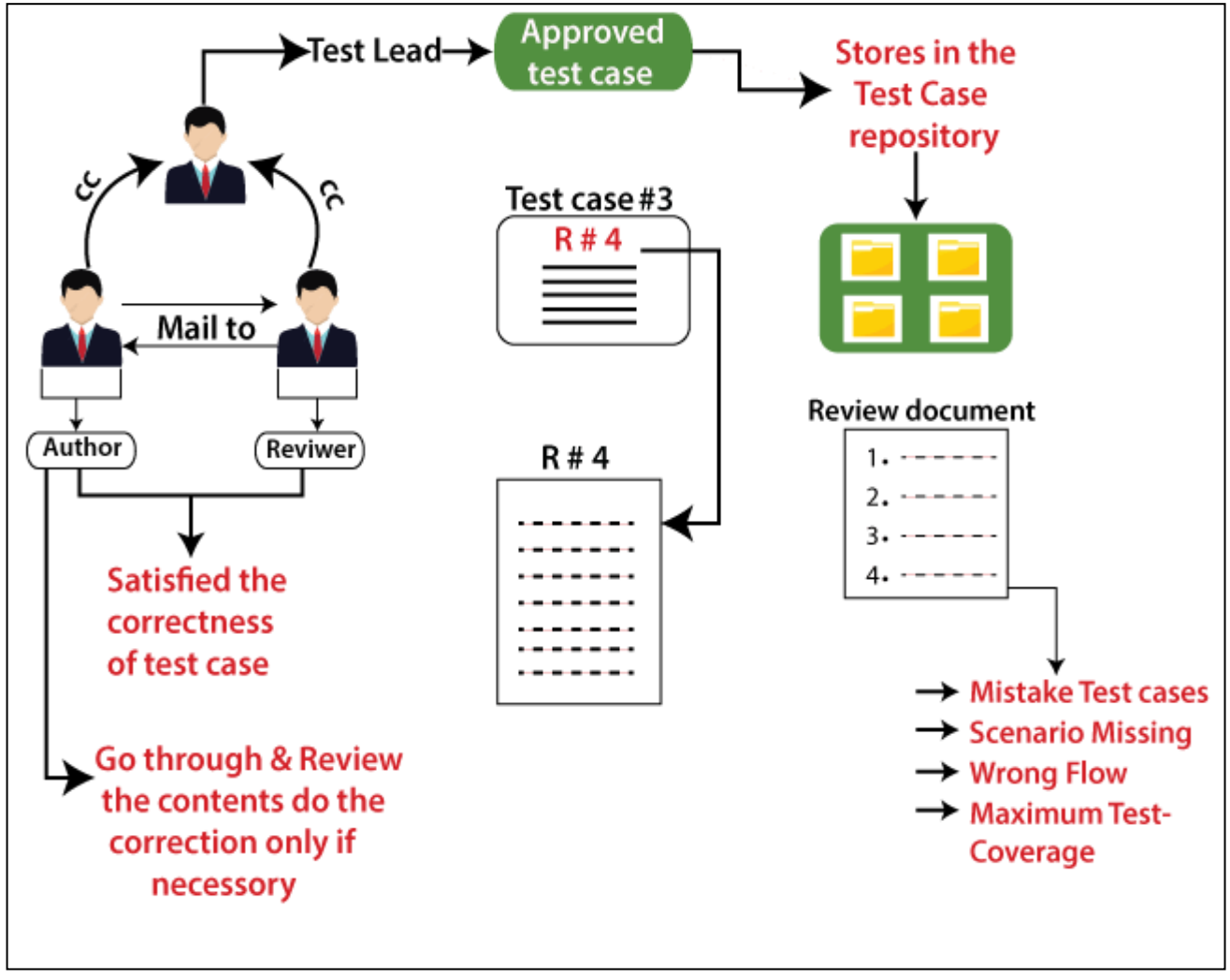
Test Case Repository

- A test case repository is a central place where all **approved test cases** are stored and managed.
- It includes test cases that contain all the key possibilities of workflow execution, thus ensuring all variations in the application and test scenarios are covered.

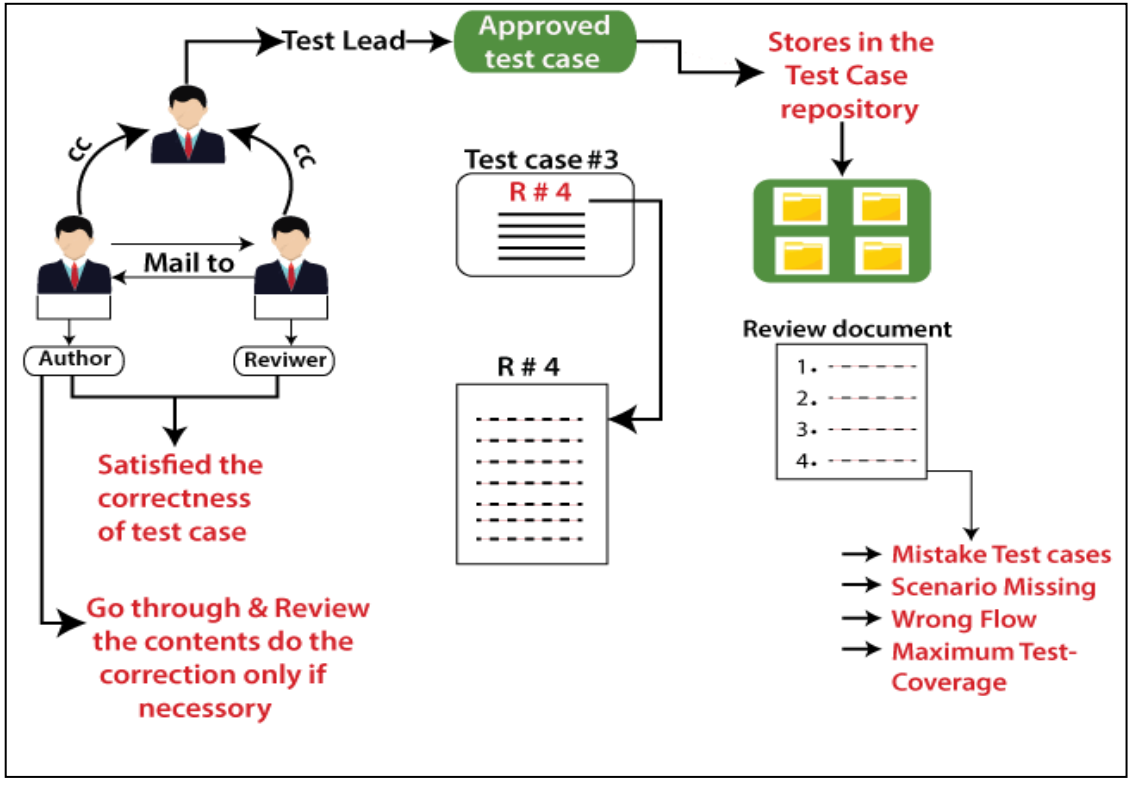


Test Case Review Process

- Test cases should be sent for review only after the test engineers finish writing them.
- This ensures the team works without interruptions.
- Once the author completes the test cases, they must be sent for review immediately.



- The reviewer checks the test case against the requirement to ensure correctness, proper flow, and maximum coverage.
- If any mistakes are found, they are documented in a review document and sent to the author.
- The author reviews the comments, makes necessary changes, and resubmits the test case for review.



- This correction process will continue until both authors, and the reviewer will satisfy.
- Once the review is successful, the reviewer sends it back to the test lead for the final approval process.

- Once the test case is reviewed, the review comments will be sent to the test case review template.

Test Case Name	Step No.	Reviewer		Author Comments	Comments
		Comments	Severity		
Cheeta-20-ICICI-AT	N.A	Pre-condition is missing	Critical	Not Fixed"No Pre-condition"	-----
	6	Used More Negative value	Minor	Fixed	-----
	11	Positive Value missing	Critical	Fixed	-----
	19	Need More Positive Value	Major	-----	-----
	20	Sentence not cleard	Minor	Not Fixed	-----

Test Execution Report

Module Name	Total No. of T.C written	Total No. of T.C Execution	Total No. of T.C Pass	Total No. of T.C Fail	Pass %	Fail %
Total						

Module name	Total No. of T.C written	Total No. of T.C Execution	Total No. of T.C pass	Total No. of T.C Fail	Pass%	Fail%
Sales	170	170	168	2	98%	2%
Amount transfer	90	90	88	2	97%	3%
Tax	127	127	127	0	100%	0%
Loans	110	110	109	1	99%	1%
Total	**497**	**497**	**492**	**5**	**97%**	**2%**

Sheet1 **Test execution report**	**Sheet2** Sales	**Sheet3** Amount transfer	**Sheet4** Tax	**Sheet5** Loans

Review

The main objectives of Software Review

(i) Detecting Problems Early

- This early detection helps save time, effort, and resources down the road.

(ii) Enhancing Quality

- They fine-tune the software to be reliable and high-quality, making sure it works well and meets user needs.

(iii) Team Collaboration

- Software reviews bring team members together to share ideas, group their expertise, and learn from each other.
- This teamwork leads to better outcomes and a stronger sense of fellowship.

(iv) Following Standards

- Software reviews ensure that the software adheres to these standards, making it consistent and aligned with best practices.

Types of Review in Software Testing

1. Software Peer Review

- Software peer review is considered as a ***collaborative effort among professionals*** to elevate the quality of their work.

a. Code Reviews

- This review process, like a team of skilled programmers checking code, ensures it follows standards, works efficiently, and is free of errors.

b. Design Reviews

- Design reviews ***evaluate the software's architecture***, and design choices.
- This guarantees efficient resource utilization, scalability, and adherence to best practices.

c. Document Reviews

- Document reviews ensure ***technical documents***, ***user guides***, ***manuals***, ***test cases*** are well-written, clear, and user-friendly.
- This careful review makes the documentation more effective and helps users understand and use the software easily.

2. Software Management Reviews

- The objective of this type of review is to ***evaluate the work status***.
- These reviews help decide the next steps in the process.

a. Project Progress Review

- Project progress reviews ***monitor and evaluate*** the project's advancement.
- These reviews provide valuable insights into project milestones, potential delays, and the need for adjustments, enabling timely decision-making and resource allocation.

b. Resource Allocation Review

- Resource allocation reviews ***examine the allocation of human resources***, ***tools***, and ***budget***.
- By ensuring efficient resource utilization, these reviews contribute to streamlined project execution and cost-effectiveness.

3. Software Audit Reviews

- Software audit reviews is consider as similar to regulatory audits in the corporate companies, ***ensuring compliance with industry standards and regulations***.
- These reviews encompass:

Regulatory Compliance Review:

- These ensure that the software aligns with specific industry regulations, legal standards, and ethical practices.

Security Audit:

- Security audit reviews assess the software's vulnerability to breaches and cyber threats.
- These reviews scrutinize the software's ability to withstand cyberattacks, safeguard sensitive data, and protect user privacy.

Inspection and Auditing

1. INSPECTIONS

- Inspections are ***formal reviews*** where moderators check documents thoroughly before a meeting.
- A meeting is then held to ***review the code*** and the ***design***.
- Inspection meetings can be held both ***physically*** and ***virtually***.
- The purpose of these meetings is to review the code and the design with everyone and to report any bugs found.
- Software inspection is divided into two types:

1. Document Inspection

- The documents produced for a given phase are inspected, further focusing on their ***quality***, ***correctness***, and ***relevance***.

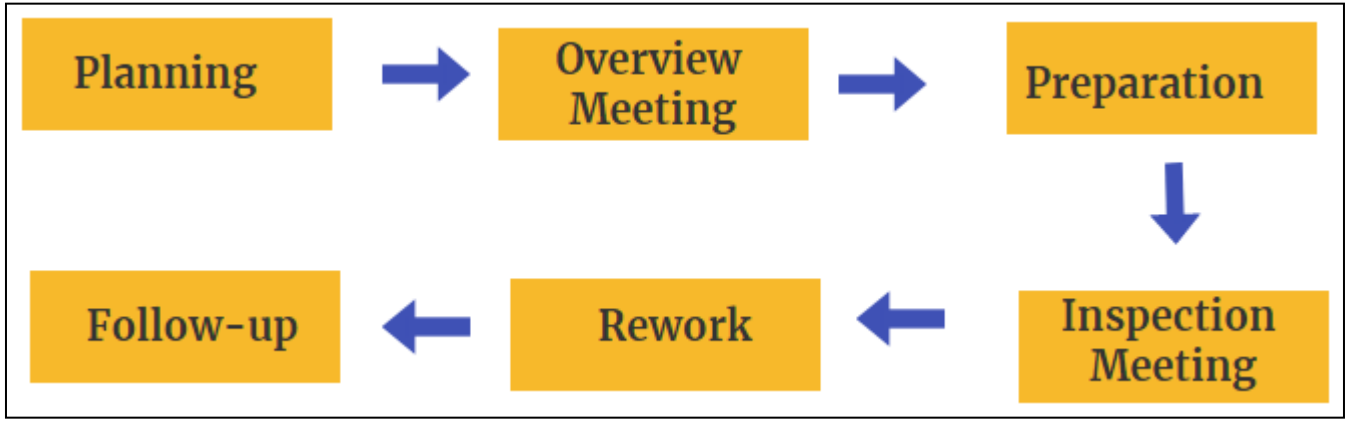
2. Code inspection

- The ***code***, program ***source files***, and ***test scenarios*** are inspected and reviewed.

A. Participants and Roles

Participants	Roles
Moderator	A facilitator who **organizes and reports** on inspection.
Author	A person (Programmer or designer) who **produces the report**.
Reader	**Present the code at an inspection meeting**, where they read the document one by one
Recorder/ Scribe	A participant who is **responsible for documenting the defects** found during the inspection process
Inspector	The inspection team member responsible for **identifying the defects**.

B. Software Inspection Process



1. Planning

- The planning phase starts with the ***selection of a group review team*** (developers, testers, and analysts).
- A **moderator** plans the activities performed during the inspection and verifies that the software entry criteria are met.

A software development company is reviewing a new mobile banking app. The team assigns a moderator and selects reviewers, including a tester and a security expert.

2. Overview Meeting

- The purpose is to provide background information about the software.
- A presentation is given to the **inspector** with some background information needed to review the software product properly.

The development team explains how the login and transaction features work in the mobile banking app, helping the reviewers understand the system before inspection.

3. Preparation

- In the ***individual preparation phase***, the **inspector** collects all the materials needed for inspection.
- **Reviewers** use checklists and past defect records to guide their review.

A security expert analyzes the login mechanism and notices a potential vulnerability in password encryption. A tester finds an issue where users cannot reset their passwords correctly.

4. Meeting

- The **moderator** conducts the meeting to ***collect and review defects***.
- The **reader** reads through the product line by line while the inspector points out the flaws.
- All issues are raised, and suggestions may be recorded.

During the review meeting, the security expert raises concerns about weak password hashing, and the tester reports a password reset bug. These issues are documented for rework.

5. Rework

- Based on meeting notes, the **author** changes the work product.

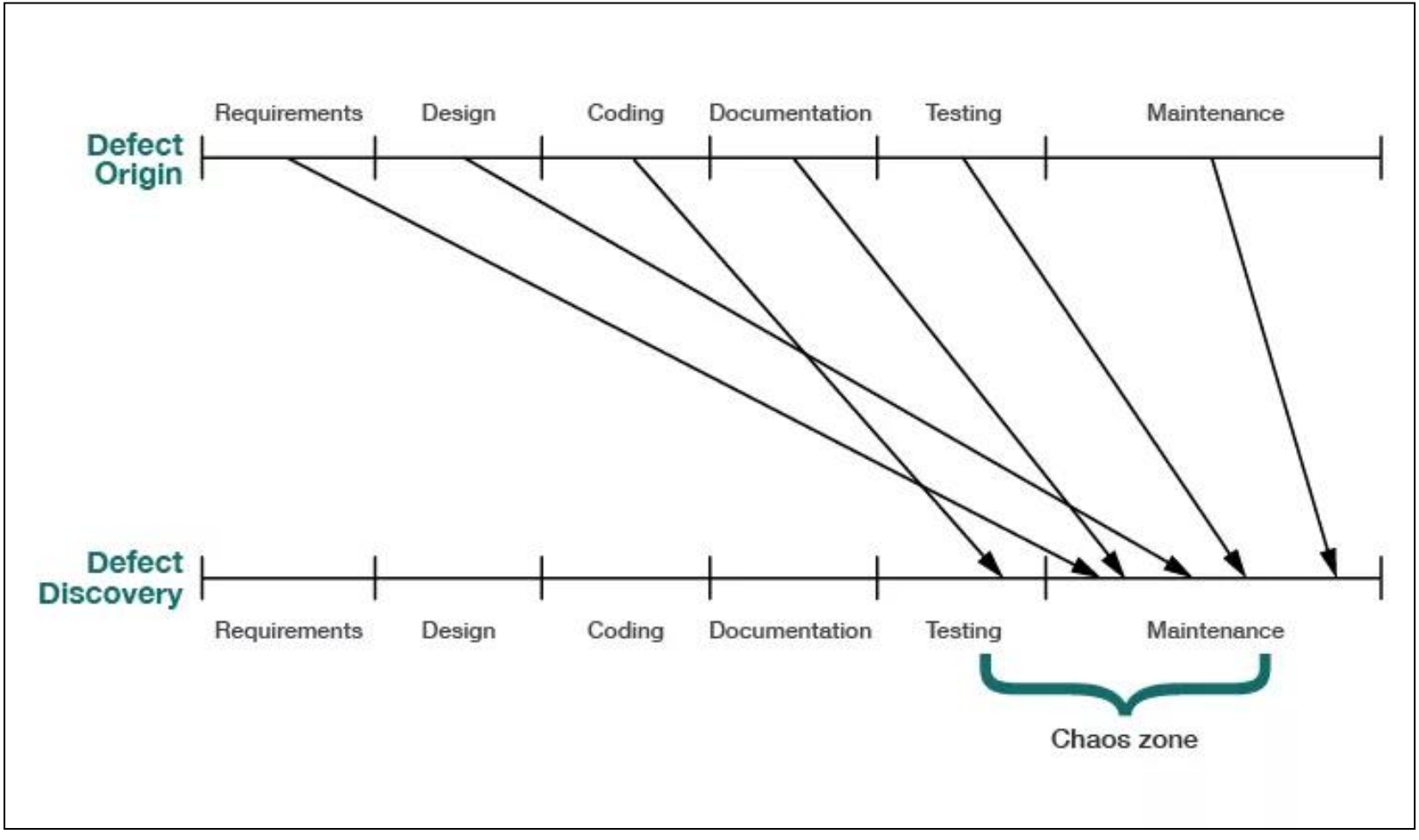
The development team fixes the password hashing issue and corrects the password reset bug in the mobile banking app.

6. Follow-up

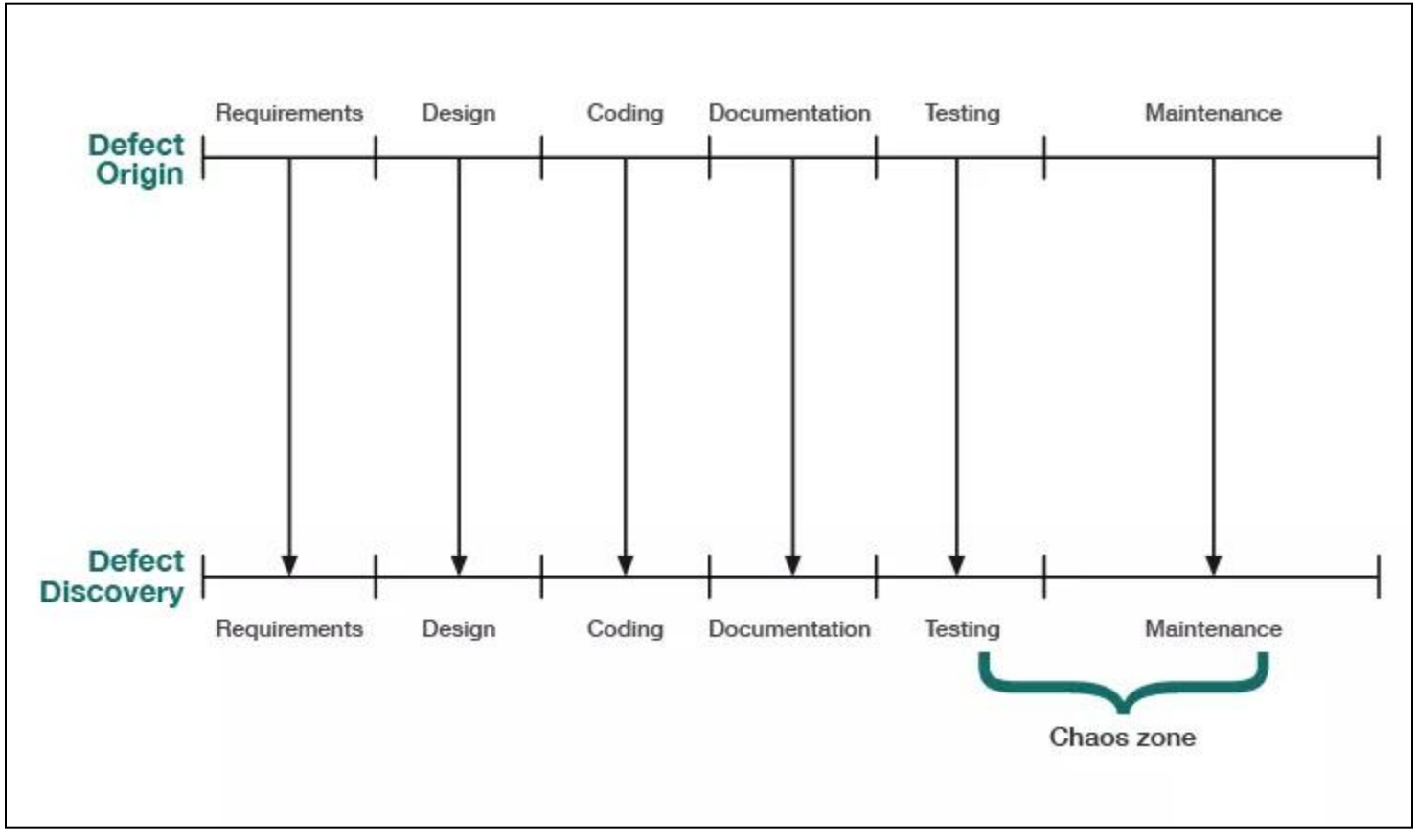
- The **moderator** checks if all defects are resolved.
- A defect summary report is created to track fixes.

The moderator verifies that the password hashing now meets security standards and the reset functionality works correctly. A summary report is prepared for documentation.

SDLC without Inspection



SDLC with Inspection



2. AUDITING

- A software audit is a ***detailed review of a software product*** to check its quality, progress, standards and regulations.
- It helps assess the product's overall health.

Types of Software Audit

(i) Audit to Verify Compliance:

- This audit checks if the ***process is within the given standards***.
- If the testing has set standards, the audit makes sure they are followed.

(ii) Audit for process improvement:

- A software audit helps find ***any needed changes*** to improve the process.
- This involves checking each step, finding problems, and fixing them.

(iii) Audit for Root Cause Analysis

- Software audit helps ***find the root cause of a problem*** using various tests.
- It focuses on specific issues that need attention and fixing.

(iv) Internal audit:

- These audits are done within the organization

(v) External audit:

- These are done by independent contractors or external agencies

- There are various ***metrics that are monitored during an audit*** to ensure that the expected outcome is being achieved.

1. Project Metrics

- **Percentage of test case execution:** Measures how many test cases have been executed

Percent of Test Case Execution **= (Number of Passed Tests + Number of Failed Tests + Number of Blocked Tests) / Number of Test Cases**

If there are 100 test cases and 80 have been executed (passed, failed, or blocked), the execution percentage is 80%.

2. Product Metrics

- **Critical defects:** Shows the number of serious issues in the product

Total Percentage of Critical Defects **= (Critical Defects / Total Defects Reported) x 100**

If a product has 50 total defects, and 10 are critical, the critical defect percentage is (10/50) × 100 = 20%.

3. People Metrics

- **Issues per reporter:** This keeps track of how many issues were reported by each reporter. It gives an idea of which defects the tester is working on, i.e., regression testing or identifying bugs.

If a tester finds 20 bugs, but another finds only 5, it helps understand who is focusing more on bug detection.

- **Tests reported by each team member:** Helps measure individual team members' contributions

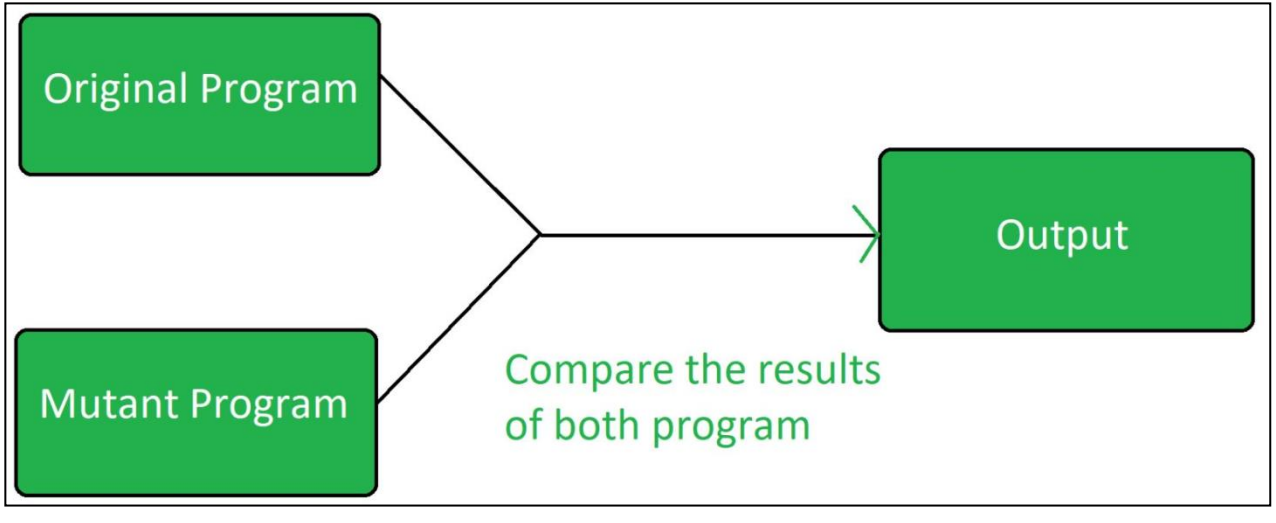
If one tester logs 30 tests and another logs 10, management can assess workload and performance.

How to perform an audit in Software Testing?

- There are some simple steps to follow while performing an audit:
  - Identify the purpose of the audit and what it hopes to find. By being specific, helps in getting optimum results and eliminates the problems efficiently
  - Examine the testing processes being done and verify the current processes against the planned and defined procedures and guidelines which were documented as a part of the testing manual prior to the testing phase
  - Once the testing process is verified, each of the test cases, test suites, test logs, defect reports, test coverage and traceability matrix are thoroughly reviewed
  - Interviewing the individuals involved at different stages in the testing process to get a better idea of the current progress

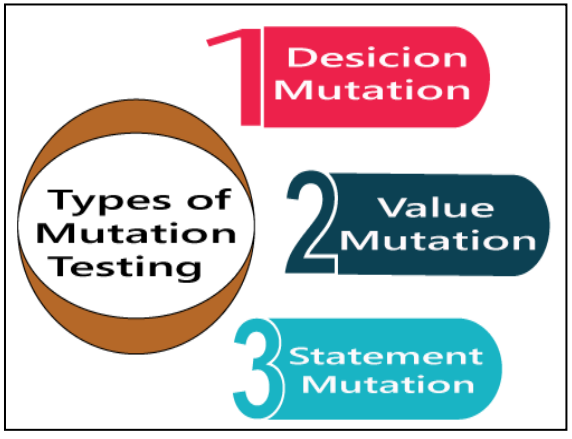
Mutation Testing

- Mutation Testing is used to evaluate and improve the quality of test cases by introducing ***small changes (mutations) in the program code*** and checking if the test cases can detect these changes.
- The primary goal of Mutation Testing is to ensure the effectiveness of test cases, making sure they can identify defects in the code.
- Mutation Testing is also called **Fault-based testing strategy** as it involves creating a fault in the program and it is a type of **White Box Testing** which is mainly used for **Unit Testing**.



- Mutation testing can be applied to ***design models***, ***specifications***, ***databases***, ***tests***, and ***XML***.
- It can be described as the ***process of rewriting the source code*** in small ways in order to remove the redundancies in the source code.
- The objective of mutation testing is:
  - *To identify pieces of code that are not tested properly.*
  - *To identify hidden defects that can't be detected using other testing methods.*
  - *To discover new kinds of errors or bugs.*
  - *To calculate the mutation score.*
  - *To study error propagation and state infection in the program.*
  - *To assess the quality of the test cases.*
- **Tools used for Mutation Testing :**
  - Judy
  - Jester
  - Jumble
  - PIT
  - MuClipse.

Types of Mutation Testing



1. Decision mutations

- In Decision Mutation, the ***logical and arithmetic operators*** used in the program are changed, which changes the overall decision-making in the program and its respective results.

Initial Code

```
if(a < b)
 c = 10;
else
 c = 20;
```

Mutant Program

```
if(a > b)
c = 10;
else
c = 20;
```

2. Value Mutation

- The ***value of constants***, ***parameters passed in the methods***, ***values used in loops*** are changed to create a mutant program.
- A small value is changed to a larger value or a larger value is changed to a smaller value

Initial Code

```
int a = 75636737;
int b = 3454;
int mult = a * b;
print(mult);
```

Mutant Code

```
int a = 75;
int b = 345466465;
int mult = a * b;
print(mult);
```

3. Statement Mutation

- In Statement Mutation, ***changes are made in the full statements of code*** in order to create a mutant program.
- Changes in the statement can be
  - *Deleting the whole statement*
  - *Changing the order of statement in code*
  - *Copy and paste the statements at some other location in code*
  - *Repeating or duplicating the few statements in the original code.*

Initial Code:

```
if (a > b)
{
print("a is greater");
}
else
{
print("b is greater");
}
```

Mutant Code:

```
if(a > b)
{
//removing the
statement
}
else
{
print("b is greater");
}
```

Initial Code:

```
if(a < b)
 c = 10;
else
 c = 20;
```

Mutant Code

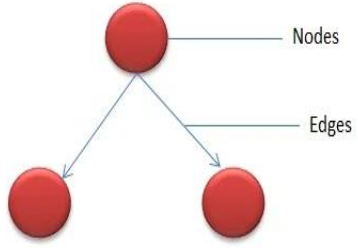
```
if(a < b)
 d = 10;
else
 d = 20;
```

How to perform mutation testing

- **Introduce Errors** : Modify the source code to create different versions, called mutants. Each mutant has a small change (error).
- **Execute Test Cases:** Run the existing test cases on both the original program and the mutant versions.
- **Compare Outputs**: Check if the mutant program produces a different output than the original.
- **Evaluate Test Cases**:
  - If the outputs ***differ***, the mutant is detected (killed), meaning the test cases are effective.
  - If the outputs ***match***, the mutant survives, indicating weak test cases that need improvement.
- **Improve Test Suite:** Modify or add test cases to ensure all mutants are detected.
- This process helps measure the effectiveness of test cases in finding defects.

CYCLOMATIC COMPLEXITY

- Cyclomatic complexity measures how complex a program is by counting the number of independent paths through its code.
- It is based on the program's control flow graph, where:
  - **Nodes** represent basic code blocks.
  - **Edges** show possible control flow between these blocks.

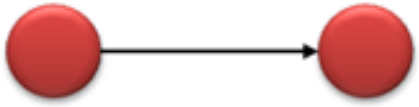


- A higher cyclomatic complexity means the code has more decision points (like loops and conditionals), making it harder to test and maintain.

Cyclomatic Complexity	Code Quality	Testability	Cost and Effort
1-10	Structured and Well Written	High	Less
10-20	Complex	Medium	Medium
20-40	Very Complex	Low	High
> 40	Highly Complex	Not at all Testable	Very High

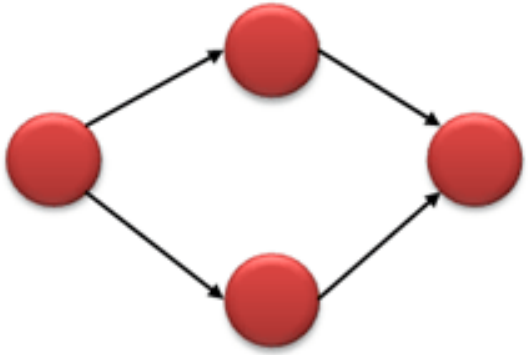
Flow graph notation for a program

Sequence



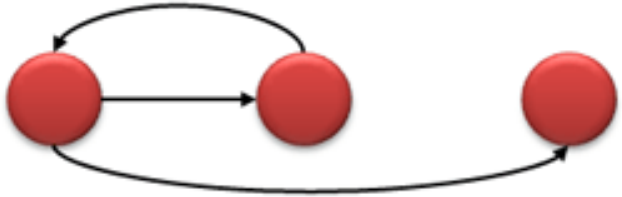
A linear sequence with only one path and no decision points.

If-then-else



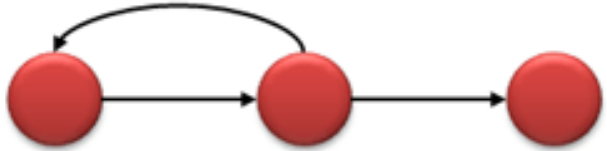
A conditional statement with three paths—true branch, false branch, and merging.

While

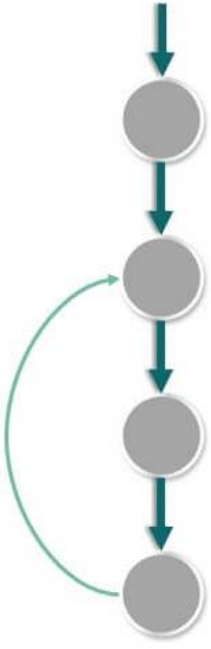


One decision point with two paths—loop continuation or exit.

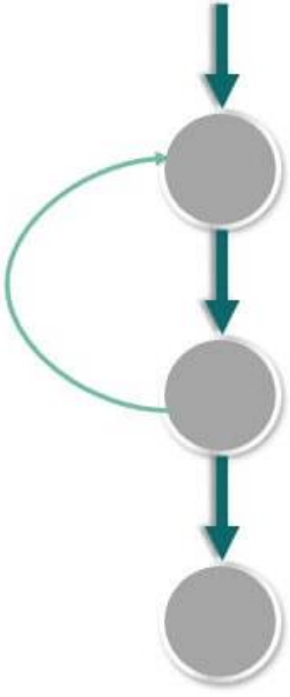
Until



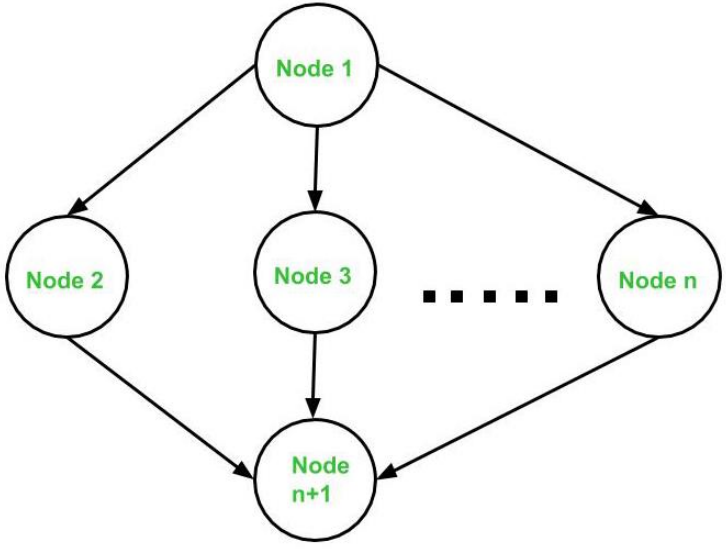
Similar to the While loop, with two paths—looping or termination.



For



Do-While



Switch Case

- Some of the metrics that can be used to measure the complexity of code include:
  - Lines of code
  - Number of dependencies
  - Test coverage
  - Ease of testing
  - Ease of reading
  - Usage of common patterns or language features
  - Good variable naming
  - Amount of duplication

Formula to Calculate the Cyclomatic Complexity

- There are 3 commonly used methods for calculating the cyclomatic complexity

Method 1: Using Edges, Nodes, and Connected Components (Graph Theory)

- The cyclomatic complexity of a control flow graph G, V(G) can be calculated using the formula given below.

$$V(G) = E - N + 2P$$

E = Number of edges or control paths in the control flow graph
N = Number of nodes in the control flow graph (if-statements, loops, or switch cases
P = Number of connected components, usually 1 for a single program.

Method 2: Using Number of Decision Points

- A simpler way to calculate cyclomatic complexity is by counting the number of decision points (conditional statements) in the program.

$$V(G) = P + 1$$

P: Number of predicate nodes (nodes containing conditions).

- Each decision point creates two branches in the control flow.

Method 3:

- Cyclomatic complexity is the number of regions in a control flow graph.

V(G) = R + 1

R = Number of regions ((areas bounded by edges)) in the graph.

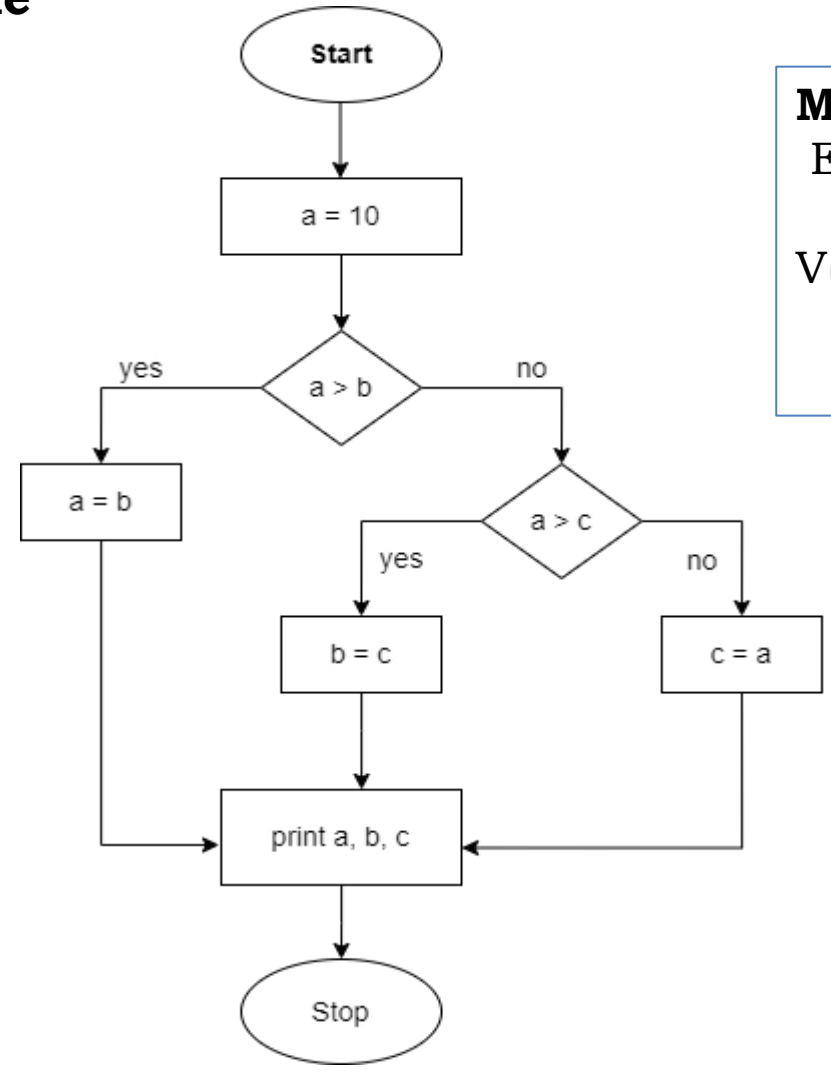
Steps that you should follow when you are calculating cyclomatic complexity and test cases design:

- ✓ **Construct a graph with nodes and edges from code.**
- ✓ **Identify the independent paths.**
- ✓ **Calculate the cyclomatic complexity**
- ✓ **Design the test cases**

1. Calculate cyclomatic complexity for the given code

```
a = 10 ;
if ( a > b ) then
    a = b ;
else
{
    if ( a > c ) then
        b = c ;
    else
        c = a ;
    end if;
}
end if;
print a
print b
print c
```

Control flow graph



Cyclomatic Complexity

Method 1:

E = 10, N = 9, P = 1

V(G) = E - N + 2*P
= 10 - 9 + 2 * 1
= **3**

Method 2:

P = 2
V(G) = P + 1
= 2 + 1
= **3**

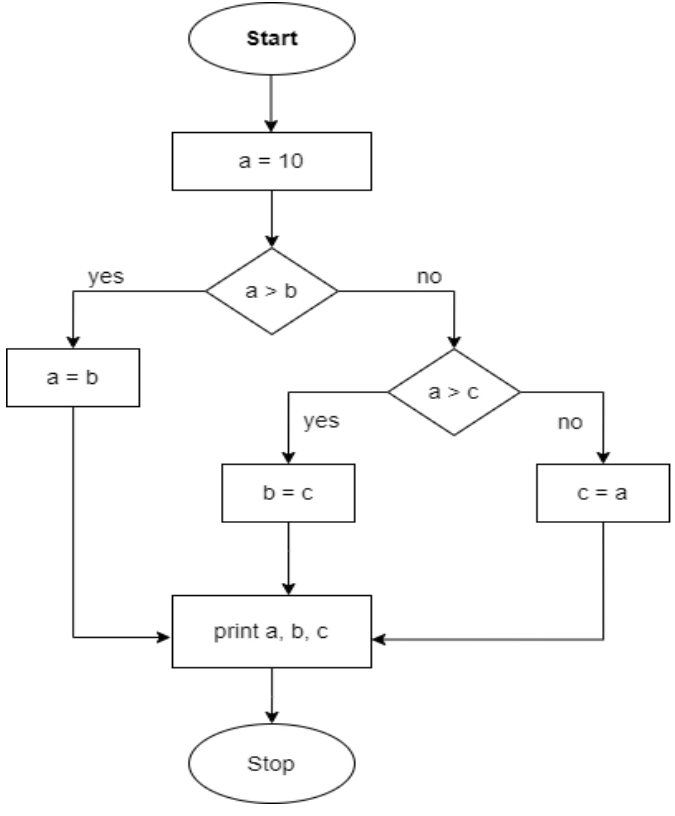
Method 3:

V(G)= R + 1
= **3**

Identify the independent paths.

- An **independent path** is any path in the program's control flow that:
  - Starts from the entry node and ends at the exit node.
  - Introduces at least one new edge (decision points like if, while, for, switch statements) that has not been traversed by previously considered paths.

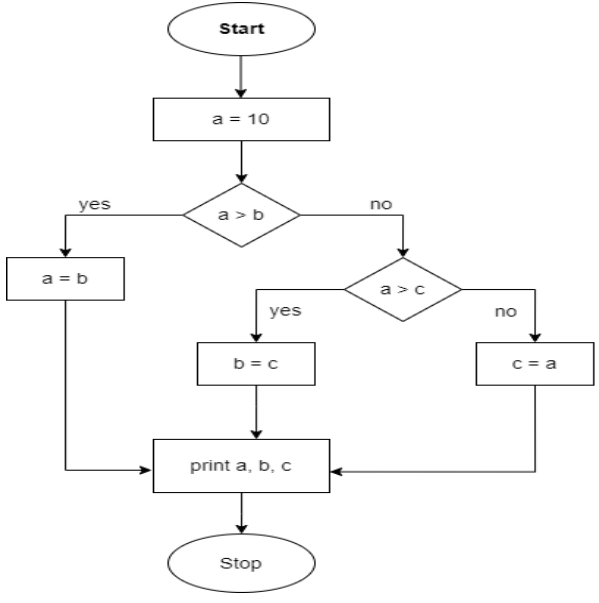
Path No.	Execution Flow
Path 1	Start → a = 10 → a > b (Yes) → a = b → Print a, b, c → Stop
Path 2	Start → a = 10 → a > b (No) → a > c (Yes) → b = c → Print a, b, c → Stop
Path 3	Start → a = 10 → a > b (No) → a > c (No) → c = a → Print a, b, c → Stop



Design the test cases

Test Case ID	Test Condition	Test Input	Expected Output	Status
TC_001	a > b is true, a > c is not checked	a = 10, b = 5, c = 8	a = 5, b = 5, c = 8	Pass/ Fail
TC_002	a > b is false, a > c is true	a = 10, b = 15, c = 8	a = 10, b = 8, c = 8	Pass/ Fail
TC_003	a > b is false, a > c is false	a = 10, b = 15, c = 20	a = 10, b = 15, c = 10	Pass/ Fail

- Cyclomatic Complexity = 3
- Three Independent Paths Identified
- Three Test Cases Created to Cover All Paths

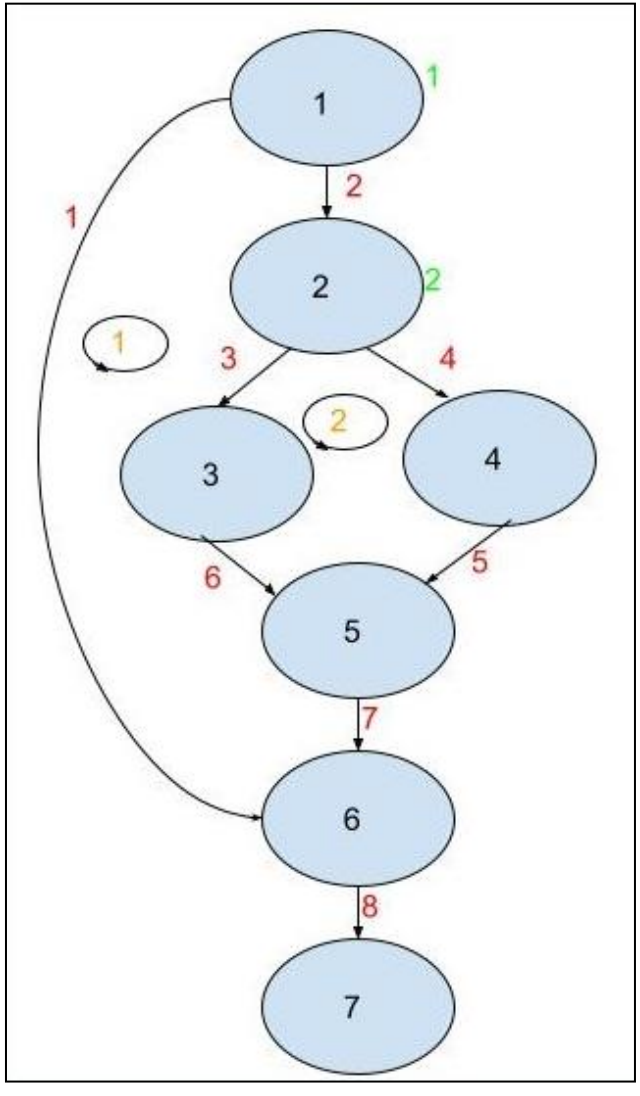


2.Calculate cyclomatic complexity for the given code

```
IF X = 300
   THEN IF Y > Z
   THEN X = Y
ELSE X = Z
   END IF
END IF
PRINT X
```

1. IF X = 300
2. THEN IF Y > Z
3. THEN X = Y
4. ELSE X = Z
5. END IF
6. END IF
7. PRINT X

Control flow graph



Cyclomatic Complexity

V(G) = E - N + 2 * P
= 8 - 7 + 2 * 1
= 3

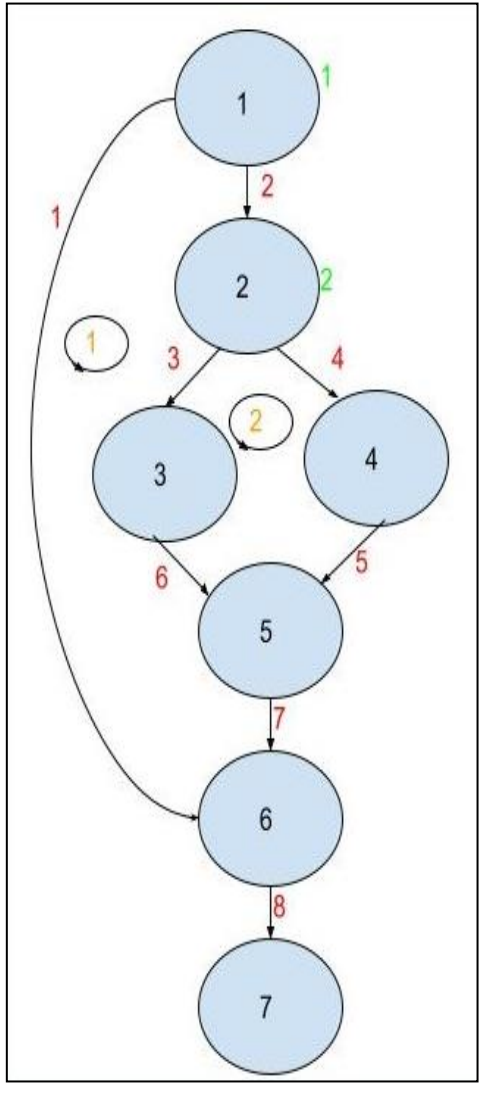
V(G) = P + 1
= 2 + 1
= 3

V(G) = R + 1
= 2 + 1
= 3

Identify the independent paths.

Path 1: 1 -> 2 -> 3 -> 5 -> 6 -> 7
Path 2: 1 -> 2 -> 4 -> 5 -> 6 -> 7
Path 3: 1 -> 6 -> 7

```
IF X = 300
   THEN IF Y > Z
        THEN X = Y
    ELSE X = Z
   END IF
END IF
PRINT X
```

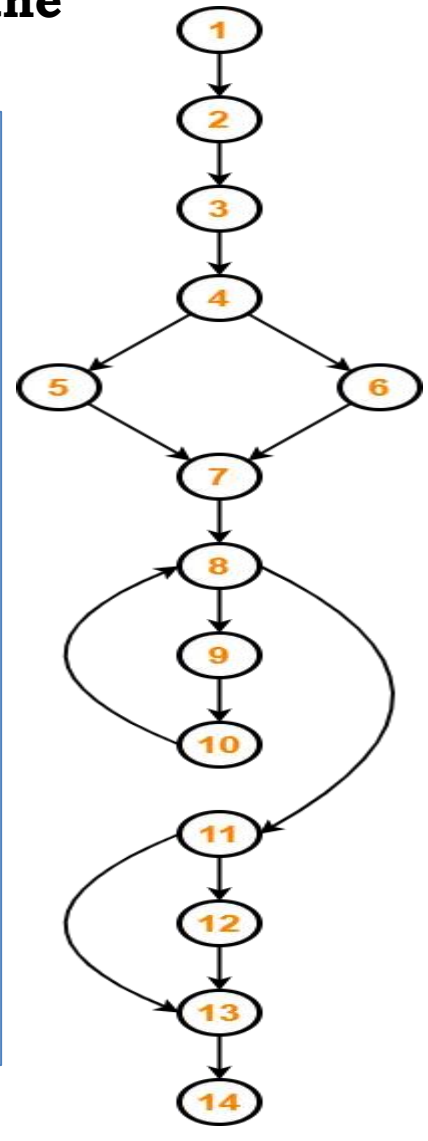


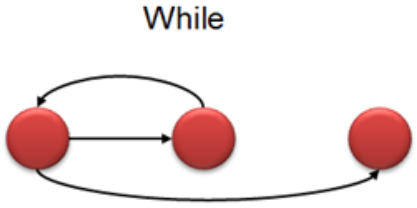
Test Case ID	Scenario	Test Input	Expected Output
TC_001	Path 1: X = 300, Y > Z	X = 300, Y = 500, Z = 200	X = 500
TC_002	Path 2: X = 300, Y ≤ Z	X = 300, Y = 100, Z = 200	X = 200
TC_003	Path 3: X ≠ 300	X = 150, Y = 400, Z = 200	X = 150

Calculate cyclomatic complexity for the given code

```
begin int x, y, power;
    float z;
    input(x, y);
     if(y<0)
          power = -y;
       else
          power = y;
    z=1;
    while(power!=0)
      {
         z=z*x;
       power=power-1;
      }
   if(y<0)
        z=1/z;
    output(z);
    end
```

```
1. begin int x, y, power;
2.     float z;
3.     input(x, y);
4.     if(y<0)
5.           power = -y;
       else
6.           power = y;
7.     z=1;
8.     while(power!=0)
       {
9.          z=z*x;
10.       power=power-1;
       }
11.    if(y<0)
12.          z=1/z;
13.    output(z);
14.    end
```





Method-01:
Cyclomatic Complexity

V(G)= E – N + 2
= 16 – 14 + 2
= 4

Method-02:
Cyclomatic Complexity

V(G) = R + 1
= 3 + 1
= 4

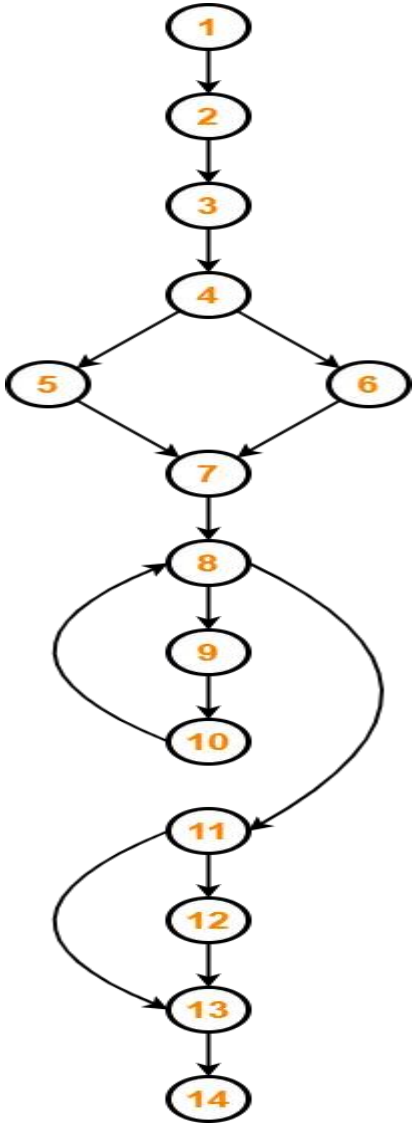
Method-03:
Cyclomatic Complexity

V(G) = P + 1
= 3 + 1
= 4

Identify the independent paths.

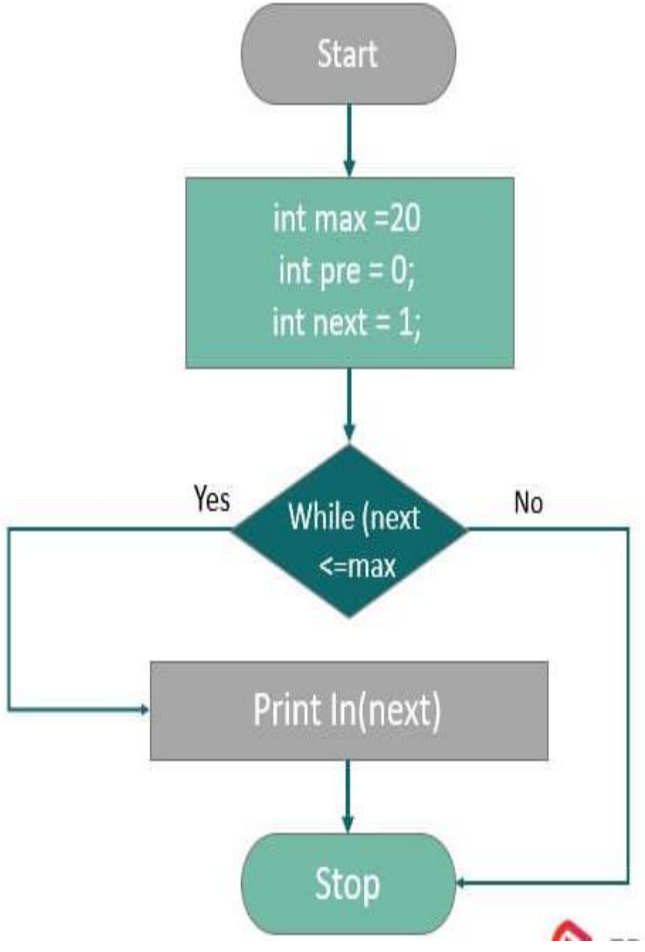
Path 1: 1-2-3-4-6-7-8-9-10-11-13-14
Path 2: 1-2-3-4-5-7-8-9-10-11-12-13-14
Path 3: 1-2-3-4-5-7-8-11-12-13-14
Path 4: 1-2-3-4-6-7-8-11-13-14

Test Case ID	Scenario	Test Input (x, y)	Expected Output (z)	Path Taken
TC1	Positive exponentiation	(2, 3)	8.0	Path 1
TC2	Negative exponentiation	(5, -2)	0.04	Path 2
TC3	Zero exponent case	(7, 0)	1.0	Path 3
TC4	Division by zero	(0, -2)	Error / Infinity	Path 4



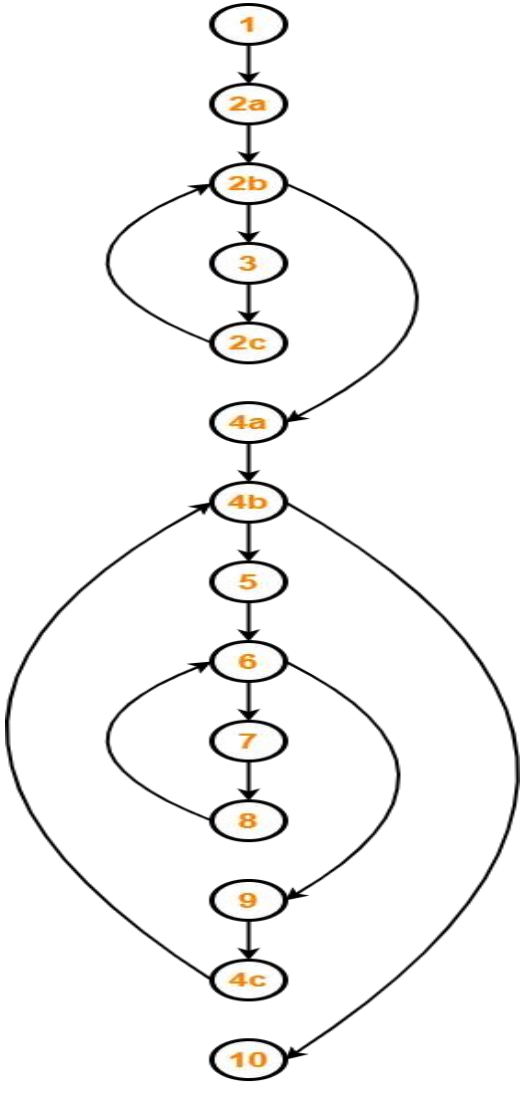
Calculate cyclomatic complexity for the given code

```
class Main {
public static void main(String[] args){
int sum;
int max = 20 ;
int pre = 0;
int next = 1;
System.out.println("The Fibonacii series is : " +pre);
while(next<= max){
System.out.println(next);
sum = pre + next;
pre = next;
next = sum;
}
}
}
```



Calculate cyclomatic complexity for the given code-

```
{ int i, j, k;
  for (i=0 ; i<=N ; i++)
  p[i] = 1;
  for (i=2 ; i<=N ; i++)
  {
     k = p[i]; j=1;
     while (a[p[j-1]] > a[k]
     {
        p[j] = p[j-1];
        j--;
  }
     p[j]=k;
}
```



Object Oriented Testing

- Object-oriented testing is a type of software testing that focuses on ***verifying the behaviour of individual objects or classes in an object-oriented system***.
- The goal of object-oriented testing is to ensure that each object or class in the system performs its functions correctly and interacts properly with other objects or classes.
- Object-oriented programming emphasises the use of objects and classes to organise and structure software, and object-oriented testing is built on these ideas.
- In object-oriented testing, the behaviour of an object or class is tested by creating test cases that simulate different scenarios or inputs that the object or class might encounter in the real world.

OBJECT-ORIENTED TESTING STRATEGIES

(i) Unit Testing in the OO Context

- In object-oriented software, units shift from ***individual modules to encapsulated classes***, which bundle data attributes and operations.
- Testing focuses on these encapsulated classes rather than isolated modules, altering the approach due to the potential overlap of operations across various classes.
- In object-oriented software, class testing resembles unit testing in traditional software.
- However, while conventional unit testing emphasizes module algorithms and data flow, class testing in OO software centers on the encapsulated operations and state behavior within each class.

(ii) Integration Testing in the OO Context

- There are two different strategies for integration testing of OO systems.

(a) Thread-based testing

- ***Integrates the set of classes required to respond to one input or event for the system***.
- Each thread is integrated and tested individually.
- Regression testing is applied to ensure that no side effects occur.

(b) Use-based testing

- Begins the construction of the system by testing those classes (called ***independent classes***) that use very few (if any) of server classes.
- After the independent classes are tested, the next layer of classes, called ***dependent classes,*** that use the independent classes are tested.
- This sequence of testing layers of dependent classes continues until the entire system is constructed.

Cluster testing

- A cluster of collaborating classes (determined by examining the CRC and object relationship model) is exercised by designing test cases that attempt to uncover errors in the collaborations.

(iii) Validation Testing in an OO Context

- At the validation or system level, the details of class connections disappear.
- Validation of OO software focuses on user-visible actions and user-recognizable outputs from the system.
- To assist in the derivation of validation tests, the tester should draw upon use cases that are part of the requirements model.
- The use case provides a scenario that has a high possibility of uncovered errors in user-interaction requirements.
- Conventional black-box testing methods can be used to drive validation tests.
- You may choose to derive test cases from the object behavior model and from an event flow diagram created as part of OOA.

OBJECT-ORIENTED TESTING METHODS

- An overall approach to OO test-case design has been suggested by Berard

  1. Each test case should be uniquely identified and explicitly associated with the class to be tested.

  2. The purpose of the test should be stated.

  3. A list of testing steps should be developed for each test and should contain:

  a. A list of specified states for the class that is to be tested

  b. A list of messages and operations that will be exercised as a consequence of the test

  c. A list of exceptions that may occur as the class is tested

  d. A list of external conditions (i.e., changes in the environment external to the software that must exist in order to properly conduct the test)

  e. Supplementary information that will aid in understanding or implementing the test

Object-Oriented Testing Levels /Techniques

(a) Fault-based testing:

- The main focus of fault-based testing is ***based on consumer specifications or code*** or both.
- Test cases are created in a way to identify all possible faults and flush them all.
- This technique finds all the defects that include ***incorrect specification and interface errors.***
- In the traditional testing model, these types of errors can be detected through functional testing.
- While Object Oriented Testing in Software Testing will require scenario-based testing.

(b) Scenario-Based Test Design

- Fault-based testing misses two main types of errors:
  (1) Incorrect specifications
  (2) Interactions among subsystems.
- Scenario-based testing focuses on simulating user actions rather than solely testing product functions.
- It involves capturing user tasks through use cases and using them as test scenarios.
- These scenarios help uncover errors in how different parts of the system interact.
- To do this effectively, test cases must be more complex and realistic compared to simple fault-based tests.
- Scenario-based testing often involves testing multiple parts of the system at once, reflecting real user behavior.

- Consider the design of scenario-based tests for a text editor by reviewing the use cases that follow

Use Case: Fix the Final Draft

Background:
It's not unusual to print the "final" draft, read it, and discover some annoying errors that weren't obvious from the on-screen image. This use case describes the sequence of events that occurs when this happens.

1. *Print the entire document.*
2. *Move around in the document, changing certain pages.*
3. *As each page is changed, it's printed.*
4. *Sometimes a series of pages is printed.*

This scenario describes two things: a test and specific user needs. The user needs are obvious: (1) a method for printing single pages and (2) a method for printing a range of pages. As far as testing goes, there is a need to test editing after printing (as well as the reverse). Therefore, you work to design tests that will uncover errors in the editing function that were caused by the printing function; that is, errors that will indicate that the two software functions are not properly independent.

Testing Web Based System

- Quality is incorporated into a Web application as a consequence of good design.
- Both reviews and testing, examine one or more of the following quality dimensions.

(i) Content

- It is evaluated at both a syntactic and semantic level.
- At the **syntactic level**, spelling, punctuation, and grammar are assessed for text-based documents.
- At a **semantic level**, correctness (of information presented), consistency (across the entire content object and related objects), and lack of ambiguity are all assessed.

(ii) Function

- Tested to ***uncover errors*** that indicate lack of conformance to customer requirements.
- Each WebApp function is assessed for correctness, instability, and general conformance to appropriate implementation standards (e.g., Java or AJAX language standards).

(iii) Structure

- Assessed to ensure that it properly ***delivers WebApp content and function***, that it is extensible, and that it can be supported as new content or functionality is added.

(iv) Usability

- Tested to ensure that ***each category of user is supported by the interface*** and can learn and apply all required navigation syntax and semantics.

(v) Navigability

- Tested to ensure that all navigation syntax and semantics are exercised to uncover any navigation errors (e.g., dead links, improper links, wrong links).

(vi) Performance

- Tested under a variety of operating conditions, configurations, and loading to ensure that the system is responsive to user interaction and handles extreme loading without unacceptable operational degradation.

(vii) Compatibility

- Tested by executing the WebApp in a variety of different host configurations on both the client and server sides.
- The intent is to find errors that are specific to a unique host configuration.

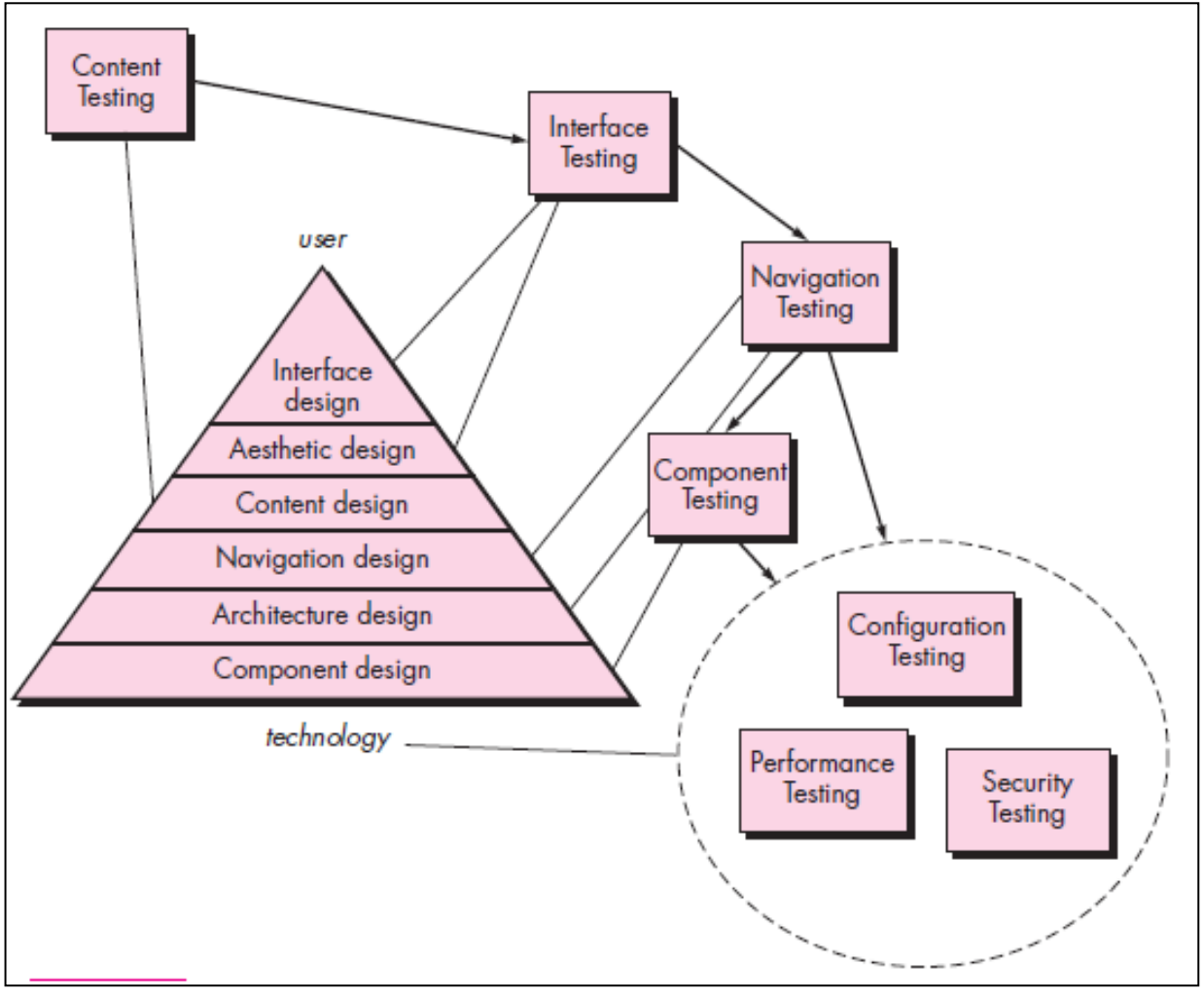
(viii) Interoperability

- Tested to ensure that the WebApp properly interfaces with other applications and/or databases.

(ix) Security

- Tested by assessing potential vulnerabilities and attempting to exploit each.
- Any successful penetration attempt is deemed a security failure.

TESTING PROCESS



1. **Functional Testing**

- Testing the ***features and operational behavior of a product*** to ensure they correspond to its specifications.
- These are the most important factors to consider while assessing the website's functionality.
  - ***Verify that all links and buttons function properly.***
  - ***Validation and submission of test forms***
  - ***Check that your website is responsive and operates on various devices and browsers.***
  - ***Ensure that your website's navigation is simple.***

Check out all the links

- Test the ***outgoing links*** from all the pages to the specific domain under test.
- Test all ***internal links***.
- Test links jumping on the same page.
- Test links are used to send emails to admin or other users from web pages.
- Test to see if there are any orphan pages.
- Finally, link checking includes checking for broken links in all the above-mentioned links.

Test forms on all pages

- Forms are an integral part of any website.
- Forms are used to receive information from users and interact with them.
  - Check all the validations in each field.
  - Check for default values in the fields.
  - Wrong inputs in the forms
  - Options to create forms, if any, form deletes a view or modify the forms.

Cookie Testing

- Cookies are small files stored on the user's machine.
- This is basically used to maintain the session – mainly the login sessions.
- Test the application by enabling or disabling the cookies in your browser options.
- Cookie Testing will include
  - ***Testing cookies*** (sessions) are deleted either when cache is cleared or when they reach their expiry.
  - ***Delete cookies*** (sessions) and test that login credentials are asked for when you next visit the site.

Validate your HTML/CSS

- Test HTML and CSS to ensure that search engines can crawl your site easily. This will include
  - Checking for Syntax Errors
  - Readable Color Schemas
  - Standard Compliance (Ensure standards such W3C, ISO, ECMA, or WS-I are followed)

2. Database Testing

- Database testing in websites is all about ***ensuring data is getting in the correct format*** in the database.
- The transactions, rollback events, speed of getting the huge list, and password encryption are all maintained in database testing.
- Performance and speed of transactions are also an integral part of here.
- Data consistency is also very important in a web application.
  - *Test data integrity.*
  - *Look for errors when updating, modifying, or performing any functionality related to the database.*
  - *Test all queries to see whether they are executing and retrieving data correctly.*

3. Usability Testing:

- Usability testing is the process by which the ***human-computer interaction characteristics of a system are measured***, and weaknesses are identified for correction.
  - Ease of learning
  - Navigation
  - Subjective user satisfaction
  - General Appearance

Test for Navigation

- Navigation means how a ***user surfs the web pages***, ***different controls like buttons, boxes***, or ***how the user uses the links on the pages*** to surf different pages.

Test the Content:

- Content should be logical and easy to understand.
- Content should be legible with no spelling or grammatical errors.
- Images if present should contain an “alt” text

4. Interface Testing

- For web testing, the ***server-side interface should be tested***.
- This can be done by verifying that the communication is done properly.
- The compatibility of the server with software, hardware, network, and database should be tested.
- **The main interfaces are:**
  - Web server and application server interface
  - Application server and Database server interface.

Application

- The application provides access via ***UI or REST/SOAP API***.

Web Server

- It is responsible for ***handling all the incoming requests*** at the backend.
- It should ensure that every incoming request is handled properly and not declined by the webserver.

Database

- Data Integrity should not be violated, and the database should provide appropriate outcomes to every query being thrown to it.
- Direct access should not be permitted, and a proper access restriction message should be returned.

5. Compatibility Testing

- It ensures the ***compatibility of applications across various devices and browsers***.

Device Compatibly

- Your application should be responsive enough to ***fit into different types of devices*** of varying sizes and shapes.
- Device compatibility testing is necessary today as everyone carries a separate device that suits their needs.

Browser Compatibility

- Application should be able to ***render itself across various browsers*** (Firefox, Chrome, Internet Explorer, Safari, etc).
- Browser compatibility testing ensures no AJAX, JavaScript, HTML, and CSS issues.

OS Compatibility:

- Test your web application on ***different operating systems*** like Windows, Unix, MAC, Linux, and Solaris with different OS flavors.

6. Performance Testing

- It tests the ***application's response time*** when put through varying load conditions.
- Performance testing can be grouped into the following categories of testing:

a. Stress Test

- It tests the maximum limit to which the web application can accept the load.
- The application is put through a load above limits, and its behavior is tested afterward.
- Stress is generally given to input fields, login, and sign-up areas.

b. Load Test

- It tests the response time of the application under varying amounts of load.
- It also measures the application server and the database's capacity.

c. Soak Test (Endurance testing)

- It measures memory utilization and CPU utilization under high load.

d. Spike Test

- Application is put through fluctuating load, and its performance is measured.
- For example, sudden decrease and increase in the number of users trying to access the application and see how the application handles these spikes.

7. Security testing

- Security Testing is vital for e-commerce website that store sensitive customer information like credit cards.
- Testing Activities will include
  - *Test unauthorized access to secure pages should not be permitted*
  - *Restricted files should not be downloadable without appropriate access*
  - *Check sessions are automatically killed after prolonged user inactivity*
  - *On use of SSL certificates, website should re-direct to encrypted SSL pages.*
- The primary reason for testing the security of a web is to identify potential vulnerabilities and subsequently repair them.
  - ❖ Network Scanning
  - ❖ Vulnerability Scanning
  - ❖ Password Cracking
  - ❖ Log Review
  - ❖ Integrity Checkers
  - ❖ Virus Detection

Types of Web Application Testing

1. Static Website Testing:

- Static websites are where ***data on the UI is directly displayed from the database***.
- There is no user interaction via the form.
- There is no contact us, comment section, or login element.
- Static Website testing is about fonts, colors, images, and user experience.

2. Dynamic Website Testing:

- Any website with a form is dynamic.
- Forms indicate user interaction.
- The data is sent from UI to the server, which stores the data in the database.
- Dynamic site testing includes UI/UX testing, API testing to ensure API is working, and API endpoints security testing.
- Finally, database testing tests that data is stored in the correct format.

3. E-Commerce Website Testing:

- E-commerce sites like Amazon and Flipkart, from where users can place their orders, are heavy on user interaction as with every button press, and location change - the data updates in a fraction of a second.
- Payment integration is also a crucial element of the e-commerce site.
- Testing all these functionalities is a part of the e-commerce site.

4. Mobile-Based Web Testing:

- Nowadays, all the sites are responsive.
- That is, the site is adjustable to the device's screen size.
- For example, the same site would have a different user experience based on screen size.
- The user experience of a site on the mobile screen, tablet screen, and laptop would be different as per the screen dimensions.
- This is a crucial element in mobile-based web testing.

Mobile App testing

- **Mobile App Testing** refers to the process of validating a mobile app (Android or iOS) for its functionality and usability before it is released publicly. Testing mobile apps help verify whether the app meets the expected technical and business requirements.