SOFTWARE TESTING

Testing:

Testing is the process if check whether the product meet customer requirement or not. Main aim is to produce good quality of product.

Its main thing is found out bugs.

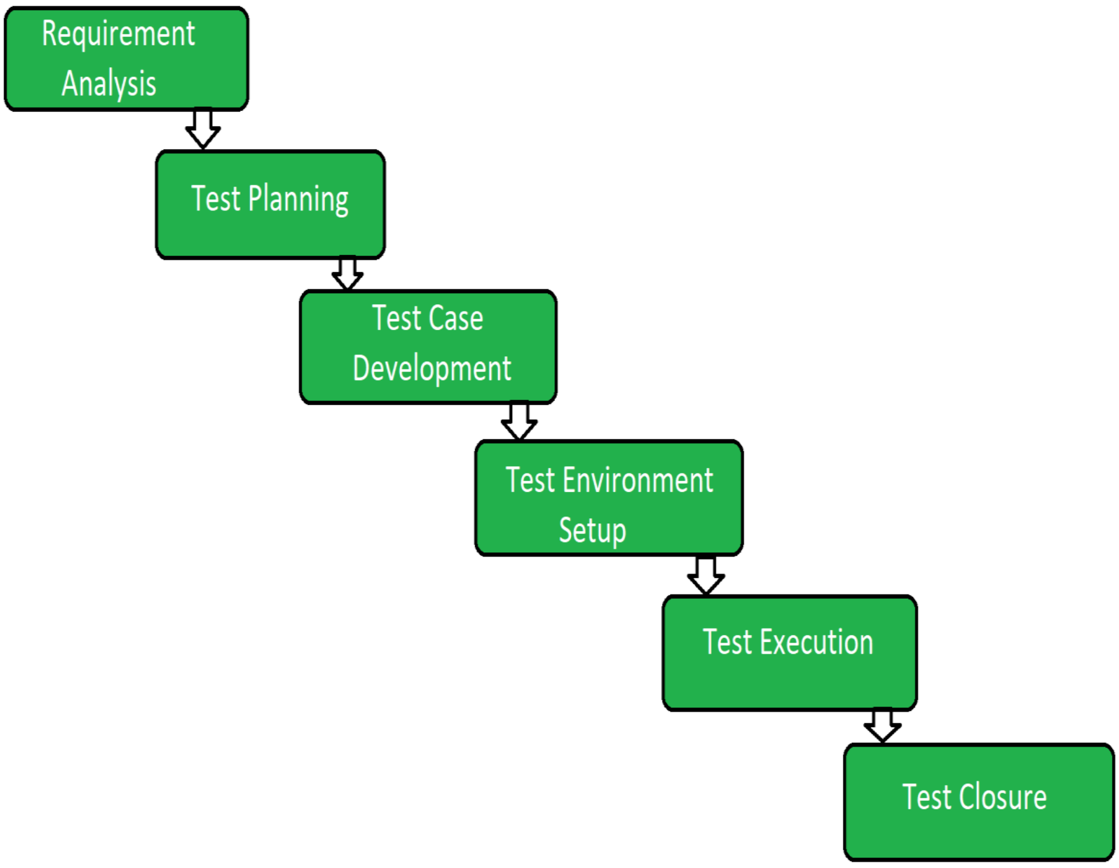
TYPES OF TESTING:

- Manual testing.
- Automation testing.

STLC (Software Testing Life Cycle) (systematic planned manner)

The main goal of the STLC is to identify and document any defects or issues in the software application as early as possible in the development process. This allows for issues to be addressed and resolved before the software is released to the public.

STLC Phases



Requirement analysis:

This is very important phase in STLC.

In this phase the QA interacts with the Business analyst, system analyst, development manager/team lead, etc or if required the QA may also interact with client to completely understand the requirement of the system.

During this phase the QA takes many important decisions like what are the testing types &techniques to be performed, feasibility for automation testing implementation etc.

ACTIVITIES: -

- Identify types of tests to be performed
- Gather detail about testing priorities and focus.
- Prepare RTM (**Requirement Traceability Matrix).**
- Automation feasibility analysis.

What is Requirement Traceability matrix?

It's used to trace the requirements to the test that are needed to verify whether the requirement are fulfilled. it's also known cross reference matrix or traceability matrix.

TYPES OF RTM

- Forward traceability.
- Backward or reverse traceability.
- Bi directional traceability(forward+backward)

Test planning:

(A test plan is detailed document that details the test strategy, objective, schedule, estimations, deadline and resources required.)

In this phase the QA/QA lead /QA manager plans for the complete testing process.

This phase is also called test strategy phase typically in the stage, a senior QA manager will determine effort and cost estimation for the project and would prepare and finalize the test plan.

This phase is very important as any small mistake in this phase can result in major issue in the project regarding time, money, effort, etc.,

Activities: -

- Prepare of test plan.
- Test tool selection.
- Test effort estimation.
- Resource planning ad determining roles and responsibilities.
- Training requirement.

TEST DESIGNING:

Creation, review & update of test cases as well as test script are done in this phase. The test scripts are done in this phase. The test cases prepared by the QA team are reviewed and approved.

Test data may also be created in this phase the QA team if test environment is available to them.

Activities: -

- ➢ Create test cases, automation scripts.
- ➢ Review & baseline test cases.

TEST Environment setup (smoke test)

Test environment is actual system /environment/ setup where the testing team be testing the application. Test environment is prepared by understanding the required system architecture, software & hardware requirements, etc.

Activities: -

- ➢ Understand the required architecture, environment set-up and prepare hardware & software requirement list for test environment.
- ➢ Setup test environment and test data

TEST EXECUTION

The test cases which were prepared earlier are executed in this phase. In this phase, the testers text the websites. Different testing techniques as well as methods are implements and executed on the software/application to break the system and find bugs.

Bugs are reported to the development team. The development team resolves the bugs and the system is retested to ensure that is bug free and ready to go live.

Activities: -

- ➢ Execute tests as per plan.
- ➢ Document test results, ad log defects for failed cases.
- ➢ Map defects to test cases in RTM.
- ➢ Retest the defects fixes.
- ➢ Track the defects to closure.

Test closure: -

When the testing team is confident that all the reported bugs are resolved and the system is read according to the client’s requirements the software testing life cycle enters the last stage i.e., Test closure stage.

In this stage, evaluations is done for the completed testing cycle, test closure reports are prepared, proper analysis and documentation is done for the major or critical bugs so that such situations can be handled efficiently and effectively in future projects, etc.

ACTIVITIES: -

- Submitting STR (software test result).
- Submitting Test Summary Report, Test Plan Document, Test cases/scripts etc.
- Sharing experiences with the team.

Difference between bug error fault, failure, defects :-

Comparison basis	BUG	DEFECT	ERROR	FAULT	FAILURE
Definition	It is an informal name specified to the defect.	The **Defect** is the difference between the actual outcomes and expected outputs.	An **Error** is a mistake made in the code; that's why we cannot execute or compile code.	The **Fault** is a state that causes the software to fail to accomplish its essential function.	If the software has lots of defects, it leads to failure or causes failure.
Raised by	The Test Engineers submit the bug.	The Testers identify the defect. And it was also solved by the developer in the development phase or stage.	The Developers and automation test engineers raise the error.	Human mistakes cause fault.	The failure finds by the manual test engineer through the development cycle.
Reasons behind	Following are reasons which may cause the bugs: Missing coding Wrong coding Extra coding	The below reason leads to the defects: Giving incorrect and wrong inputs. Dilemmas and errors in the outside behavior and inside structure and design.	The reasons for having an error are as follows: Errors in the code. The Mistake of some values. If a developer is unable to compile or run a program successfully. Confusions and issues in programming. Invalid login, loop, and syntax. Inconsistency between actual and expected outcomes.	The reasons behind the fault are as follows: A Fault may occur by an improper step in the initial stage, process, or data definition. Inconsistency or issue in the program. An irregularity or loophole in	Following are some of the most important reason behind the failure: Environmental condition System usage Users Human error

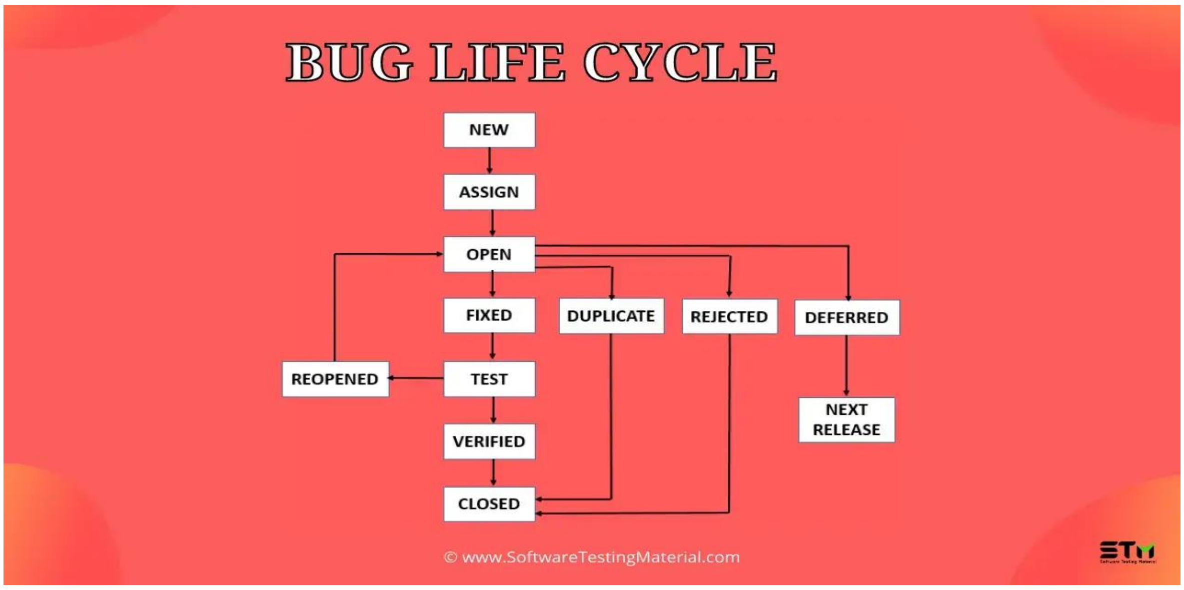
		An error in coding or logic affects the software and causes it to breakdown or the failure.	Blunders in design or requirement actions. Misperception in understanding the requirements of the application	the software that leads the software to perform improperly	

BUG LIFE CYCLE OR DEFECT LIFE CYCLE:

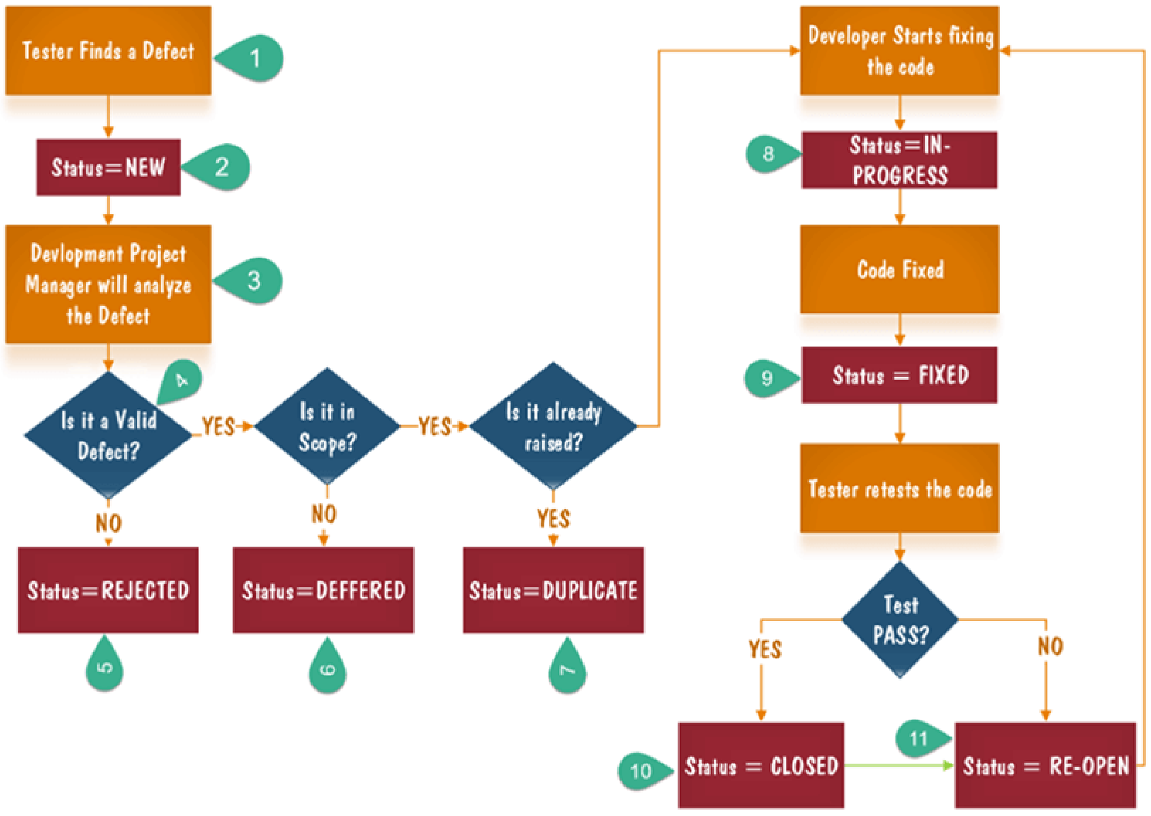
Bug Life Cycle starts with an unintentional software bug/behaviour and end when the assigned developer fixes the bug a bug when found should be communicated and assigned to a developer that can fix it.

Defect States Workflow

- **New:** When a new defect is logged and posted for the first time. It is assigned a status as NEW.
- **Assigned:** Once the bug is posted by the tester, the lead of the tester approves the bug and assigns the bug to the developer team
- **Open**: The developer starts analyzing and works on the defect fix
- **Fixed**: When a developer makes a necessary code change and verifies the change, he or she can make bug status as "Fixed."
- **Pending retest**: Once the defect is fixed the developer gives a particular code for retesting the code to the tester. Since the software testing remains pending from the testers end, the status assigned is "pending retest."
- **Retest**: Tester does the retesting of the code at this stage to check whether the defect is fixed by the developer or not and changes the status to "Re-test."



Defect/Bug Life Cycle Explained



EXPLANATION:

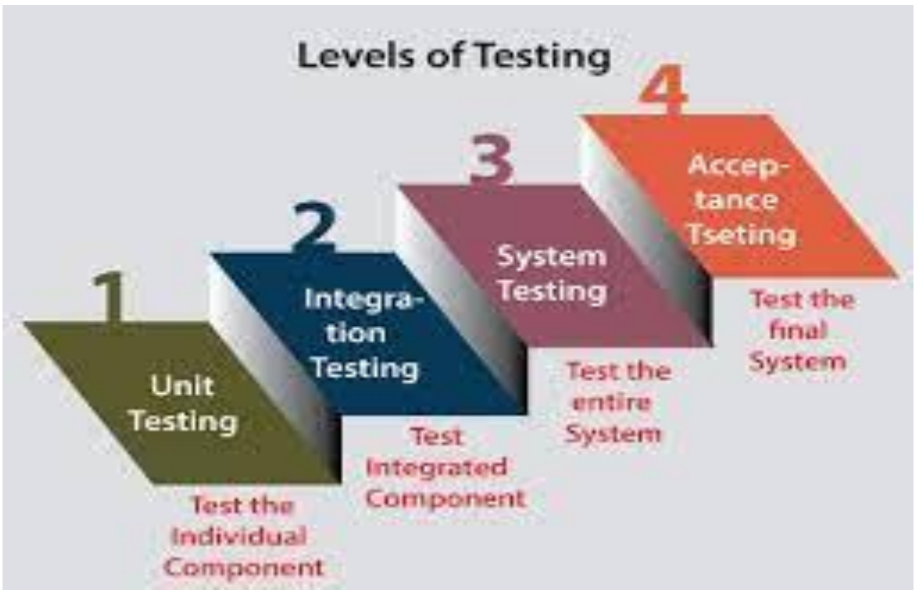
1. Tester finds the defect
2. Status assigned to defect- New

3. A defect is forwarded to Project Manager for analyze
4. Project Manager decides whether a defect is valid
5. Here the defect is not valid- a status is given “Rejected.”
6. So, project manager assigns a status **rejected**. If the defect is not rejected then the next step is to check whether it is in scope. Suppose we have another function- email functionality for the same application, and you find a problem with that. But it is not a part of the current release when such defects are assigned as a **postponed or deferred** status.
7. Next, the manager verifies whether a similar defect was raised earlier. If yes defect is assigned a status **duplicate**.
8. If no the defect is assigned to the developer who starts fixing the code. During this stage, the defect is assigned a status **in- progress.**
9. Once the code is fixed. A defect is assigned a status **fixed**
10. Next, the tester will re-test the code. In case, the Test Case passes the defect is **closed.** If the test cases fail again, the defect is **re-opened** and assigned to the developer.
11. Consider a situation where during the 1st release of Flight Reservation a defect was found in Fax order that was fixed and assigned a status closed. During the second upgrade release the same defect again re-surfaced. In such cases, a closed defect will be **re-opened.**

LEVELS OF TESTING

- **Unit testing**
- **Integration testing**
- **System testing**
- **Acceptance testing**

Block diagram:



Unit testing:

- ❖ Unit testing is when every module of the application gets tested respectively.
- ❖ Unit testing is done by the developer himself. After he has written code for a feature, he will ensure it is working fine.
- ❖ Unit tests are the smallest testable component of the application.
- ❖ Nowadays we have Junit, Pytest, and TestNg frameworks for unit testing the application.

Integration testing:

- ❖ Integration testing is a testing technique where two or more independent components are tested together.
- ❖ Integration testing is done by the developer. Here test cases are written to ensure the data flowing between them is correct.
- ❖ For example, testing the signup form where UI validations are correct, data reaching API, and getting stored are all validated.
- ❖ Integration testing is done when the application is still developing to find bugs early on in the development process.

System Testing

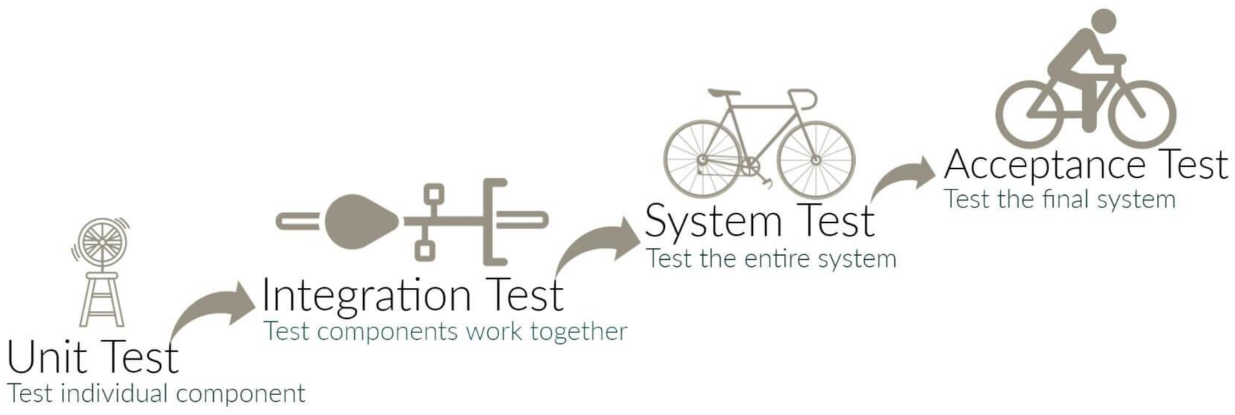
- ❖ System testing is done by the tester where the entire application is tested as a single unit.
- ❖ Hence, system testing test cases are also performance test cases, load testing, and stress testing test cases.

- System testing is done to find the errors which might have been overlooked during unit or integration testing.
- System testing evaluates both functional and non-functional test cases.

Acceptance Testing

- Acceptance testing is done by the client where he evaluates whether the product is made by the requirement he listed out.
- Acceptance testing is done at the UAT server where a well-tested product is deployed by the team for the client's reference so he can track ongoing changes in the project
- There is a defined acceptance criterion that is laid at the time of requirement listing so that the client can validate that the product is meeting the acceptance criteria.
- Once the client completes acceptance testing the product goes to production where users can use the final application.

Example:



CHARACTERISTICS OF TESTING:

1. **Functional testing**
2. **Nonfunctional testing**

Function testing:

Functional testing is a type of testing which verifies that each **function** of the software application operates in conformance with the requirement specification. This testing mainly involves black box testing, and it is not concerned about the source code of the application.

Every functionality of the system is tested by providing appropriate input, verifying the output and comparing the actual results with the expected results. This testing involves checking of User Interface, APIs, Database, security, client/ server applications and functionality of the Application Under Test. The testing can be done either manually or using automation

Nonfunctional testing:

Non-functional testing is a type of testing to check non-functional aspects (performance, usability, reliability, etc.) of a software application. It is explicitly designed to test the readiness of a system as per nonfunctional parameters which are never addressed by functional testing.

A good example of non-functional test would be to check how many people can simultaneously login into a software.

Non-functional testing is equally important as functional testing and affects client satisfaction.

Difference between functional and nonfunctional testing

PARAMETERS	**FUNCTIONAL**	**NONFUNCTIONAL**
EXECUTION	It is performed before non-functional testing.	It is performed after the functional testing.
FOCUS AREA	It is based on customer's requirements.	It focusses on customer's expectation
REQUIREMENT	It is easy to define functional requirements.	It is difficult to define the requirements for non-functional testing.

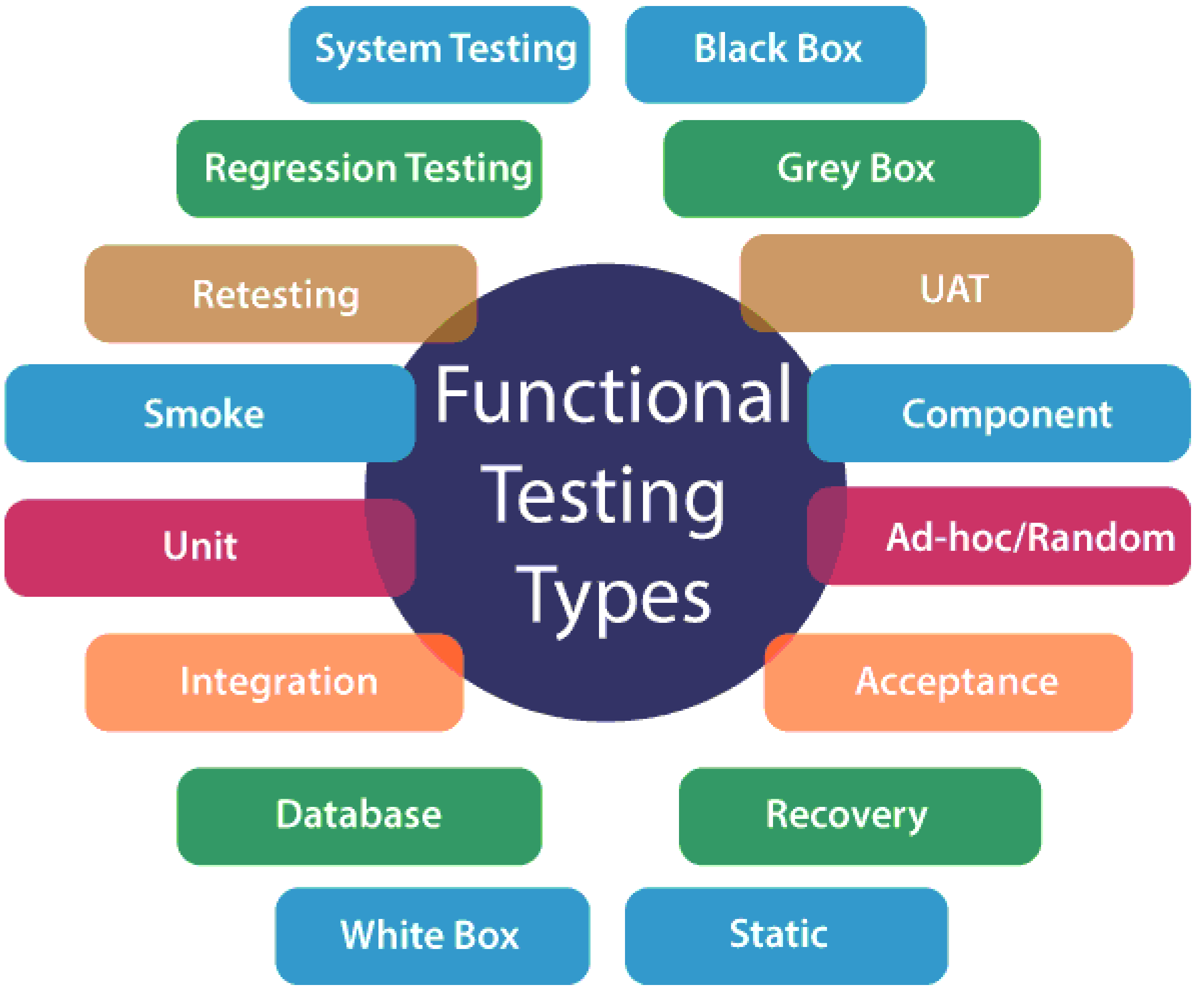
USAGE	Helps to validate the behaviour of the application.	Helps to validate the performance of the application.
OBJECTIVES	Carried out to validate software actions.	It is done to validate the performance of the software.
MANUAL TESTING	Functional testing is easy to execute by manual testing.	It's very hard to perform non-functional testing manually.
FUNCTIONALITY	It describes what the product does.	It describes how the product works.
Example Test Case	Check login functionality.	The dashboard should load in 2 seconds.
TESTING TYPES	Examples of Functional Testing Types • Unit testing • Smoke testing • User Acceptance • Integration Testing • Regression testing • Localization • Globalization • Interoperability	Examples of Non-functional Testing Types • Performance Testing • Volume Testing • Scalability • Usability Testing • Load Testing • Stress Testing • Compliance Testing • Portability Testing • Disaster Recover Testing

Explain the types of functional testing.

The main objective of functional testing is to test the functionality of the component.

Functional testing is divided into multiple parts.

Here are the following types of functional testing.



Unit Testing: Unit testing is a type of software testing, where the individual unit or component of the software tested. Unit testing, examine the different part of the application, by unit testing functional testing also done, because unit testing ensures each module is working correctly.

The developer does unit testing. Unit testing is done in the development phase of the application.

Smoke Testing: Functional testing by smoke testing. Smoke testing includes only the basic (feature) functionality of the system. Smoke testing is known as

"***Build Verification Testing***." Smoke testing aims to ensure that the most important function work.

For example, Smoke testing verifies that the application launches successfully will check that GUI is responsive.

Sanity Testing: Sanity testing involves the entire high-level business scenario is working correctly. Sanity testing is done to check the functionality/bugs fixed. Sanity testing is little advance than smoke testing.

For example, login is working fine; all the buttons are working correctly; after clicking on the button navigation of the page is done or not.

Regression Testing: This type of testing concentrate to make sure that the code changes should not side effect the existing functionality of the system. Regression testing specifies when bug arises in the system after fixing the bug, regression testing concentrate on that all parts are working or not. Regression testing focuses on is there any impact on the system.

Integration Testing: Integration testing combined individual units and tested as a group. The purpose of this testing is to expose the faults in the interaction between the integrated units.

Developers and testers perform integration testing.

White box testing: White box testing is known as Clear Box testing, code-based testing, structural testing, extensive testing, and glass box testing, transparent box testing. It is a software testing method in which the internal structure/design/ implementation tested known to the tester.

The white box testing needs the analysis of the internal structure of the component or system.

Black box testing: It is also known as behavioral testing. In this testing, the internal structure/ design/ implementation not known to the tester. This type of testing is functional testing. Why we called this type of testing is black-box testing, in this testing tester, can't see the internal code.

For example, A tester without the knowledge of the internal structures of a website tests the web pages by using the web browser providing input and verifying the output against the expected outcome.

User acceptance testing: It is a type of testing performed by the client to certify the system according to requirement. The final phase of testing is user acceptance testing before releasing the software to the market or production environment. UAT is a kind of black-box testing where two or more end-users will involve.

Retesting: Retesting is a type of testing performed to check the test cases that were unsuccessful in the final execution are successfully pass after the defects fixed. Usually, tester assigns the bug when they find it while testing the product or its component. The bug allocated to a developer, and he fixes it. After fixing, the bug is assigned to a tester for its verification. This testing is known as retesting.

Database Testing: Database testing is a type of testing which checks the schema, tables, triggers, etc. of the database under test. Database testing may involve creating complex queries to load/stress test the database and check its responsiveness. It checks the data integrity and consistency.

Example: let us consider a banking application whereby a user makes a transaction. Now from database testing following, things are important. They are:

- Application store the transaction information in the application database and displays them correctly to the user.
- No information lost in this process
- The application does not keep partially performed or aborted operation information.
- The user information is not allowed individuals to access by the

Ad-hoc testing: Ad-hoc testing is an informal testing type whose aim is to break the system. This type of software testing is unplanned activity. It does not follow any test design to create the test cases. Ad-hoc testing is done randomly on any part of the application; it does not support any structured way of testing.

Recovery Testing: Recovery testing is used to define how well an application can recover from crashes, hardware failure, and other problems. The purpose of recovery testing is to verify the system's ability to recover from testing points of failure.

Static Testing: Static testing is a software testing technique by which we can check the defects in software without actually executing it. Static testing is done to avoid errors in the early stage of the development as it is easier to find failure in the early stages. Static testing used to detect the mistakes that may not found in dynamic testing.

Why we use static testing?

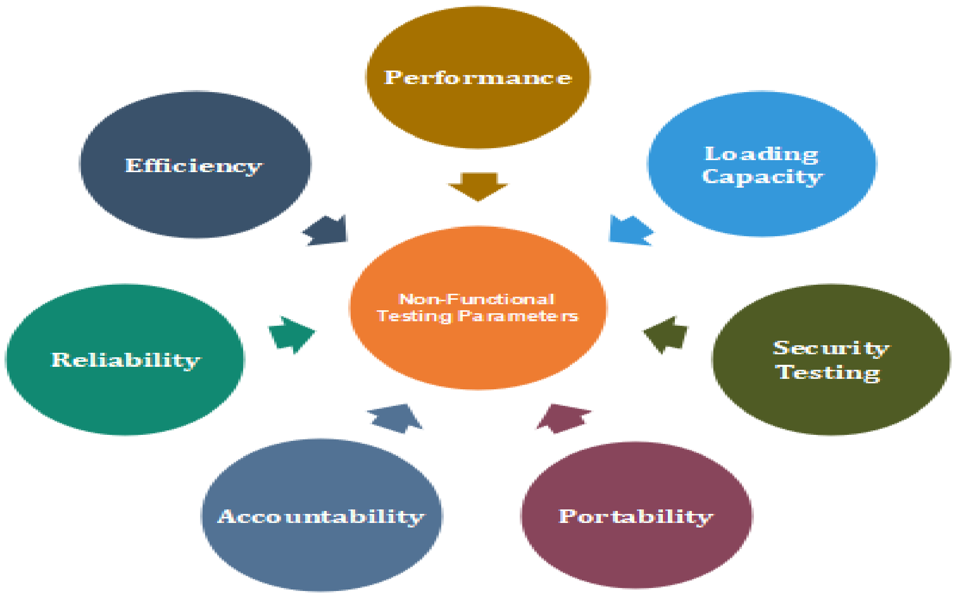
Static testing helps to find the error in the early stages. With the help of static testing, this will reduce the development timescales. It reduces the testing cost and time. Static testing also used for development productivity.

Component Testing: Component Testing is also a type of software testing in which testing is performed on each component separately without integrating with other parts. Component testing is also a type of black-box testing. Component testing also referred to as Unit testing, program testing, or module testing.

Grey Box Testing: Grey Box Testing defined as a combination of both white box and black-box testing. Grey Box testing is a testing technique which performed with limited information about the internal functionality of the system.

Black Box
+
White Box
=
Grey Box

Parameters to be tested under Non-Functional Testing



Performance Testing

Performance Testing eliminates the reason behind the slow and limited performance of the software. Reading speed of the software should be as fast as possible.

For Performance Testing, a well-structured and clear specification about expected speed must be defined. Otherwise, the outcome of the test (Success or Failure) will not be obvious.

Load Testing

Load testing involves testing the system's loading capacity. Loading capacity means more and more people can work on the system simultaneously.

Security Testing

Security testing is used to detect the security flaws of the software application. The testing is done via investigating system architecture and the mindset of an attacker. Test cases are conducted by finding areas of code where an attack is most likely to happen.

Portability Testing

The portability testing of the software is used to verify whether the system can run on different operating systems without occurring any bug. This test also tests the working of software when there is a same operating system but different hardware.

Accountability Testing

Accountability test is done to check whether the system is operating correctly or not. A function should give the same result for which it has been created. If the system gives expected output, it gets passed in the test otherwise failed.

Reliability Testing

Reliability test assumes that whether the software system is running without fail under specified conditions or not. The system must be run for a specific time and number of processes. If the system is failed under these specified conditions, reliability test will be failed.

Efficiency Testing

Efficiency test examines the number of resources needed to develop a software system, and how many of these were used. It also includes the test of these three points.

- Customer's requirements must be satisfied by the software system.
- A software system should achieve customer specifications.
- Enough efforts should be made to develop a software system.

Advantages of Non-functional testing

- It provides a higher level of security. Security is a fundamental feature due to which system is protected from cyber-attacks.
- It ensures the loading capability of the system so that any number of users can use it simultaneously.
- It improves the performance of the system.
- Test cases are never changed so do not need to write them more than once.
- Overall time consumption is less as compared to other testing processes.

Disadvantages of Non-Functional Testing

- Every time the software is updated, non-functional tests are performed again.
- Due to software updates, people have to pay to re-examine the software; thus software becomes very expensive.

TEST PLAN CASE:

A **Test Plan** is a detailed document that describes the test strategy, objectives, schedule, estimation, deliverables, and resources required (human resources, software, and hardware) to perform testing for a software product. Test Plan helps us determine the effort needed to validate the quality of the application under test. The test plan serves as a blueprint to conduct software testing activities as a defined process, which is minutely monitored and controlled by the test manager.

“Test Plan is A document describing the scope, approach, resources, and schedule of intended test activities.”

What is the Importance of Test Plan?

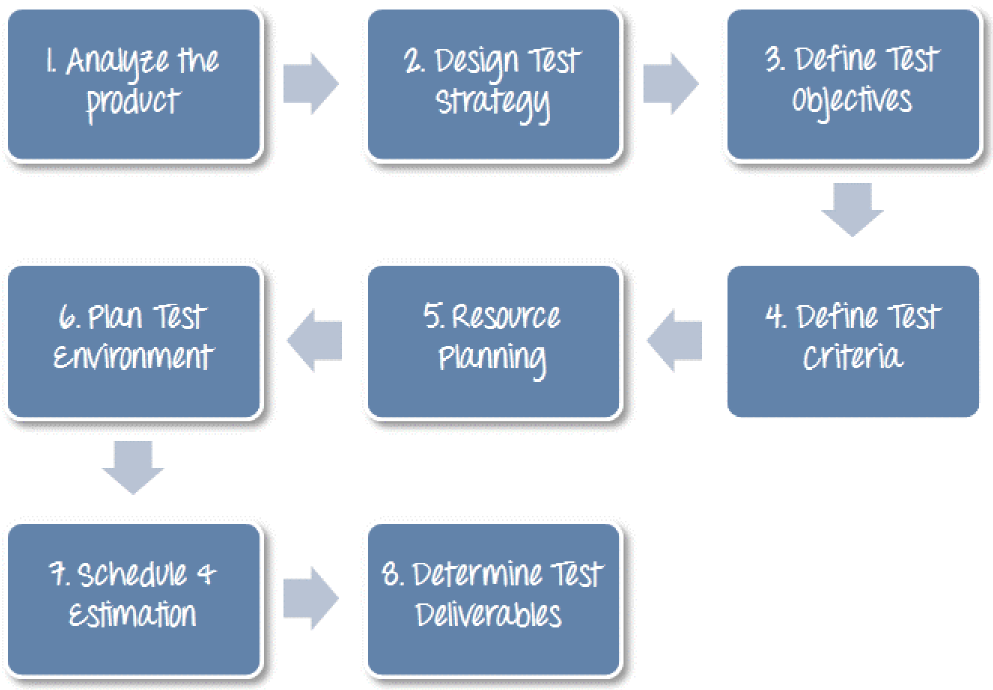
Making Test Plan document has multiple benefits

- Help people outside the test team such as developers, business managers, customers **understand** the details of testing.
- Test Plan **guides** our thinking. It is like a rule book, which needs to be followed.
- Important aspects like test estimation, test scope, Test Strategy are **documented** in Test Plan, so it can be reviewed by Management Team and re-used for other projects.

How to write a Test Plan

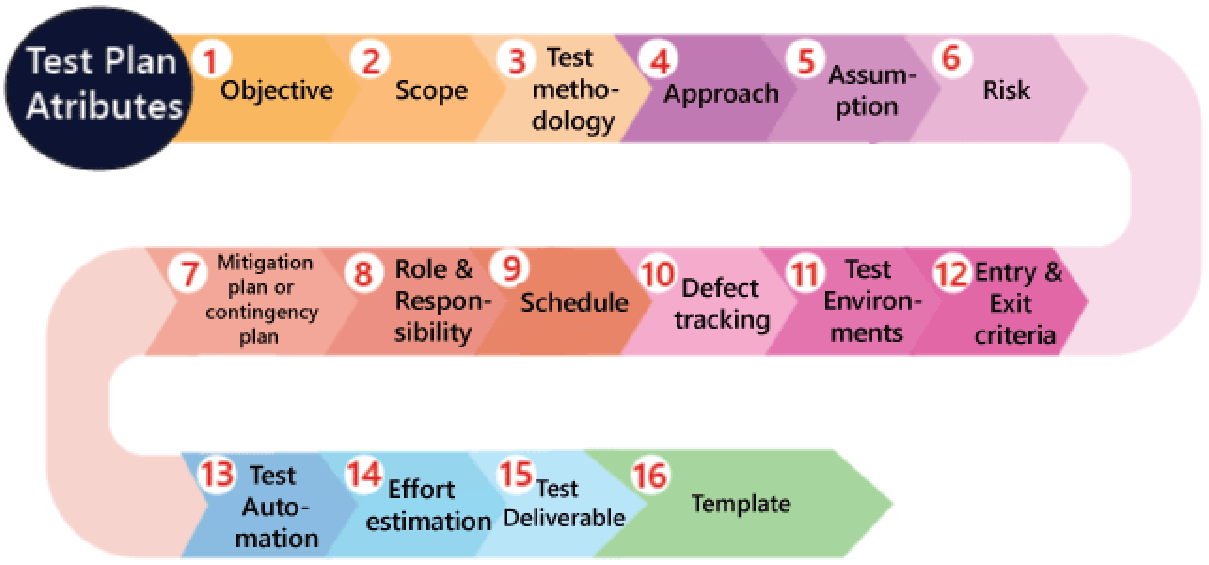
You already know that making a **Test Plan** is the most important task of Test Management Process. Follow the seven steps below to create a test plan as per IEEE 829

1. Analyse the product
2. Design the Test Strategy
3. Define the Test Objectives
4. Define Test Criteria
5. Resource Planning
6. Plan Test Environment
7. Schedule & Estimation
8. Determine Test Deliverables



Test plan components or attributes

The test plan consists of various parts, which help us to derive the entire testing activity.



Objectives: It consists of information about modules, features, test data etc., which indicate the aim of the application means the application behaviour, goal, etc.

Scope: It contains information that needs to be tested with respective of an application. The Scope can be further divided into two parts:

- In scope
- Out scope

In scope: These are the modules that need to be tested rigorously (in-detail)

Out scope: These are the modules, which need not be tested rigorously.

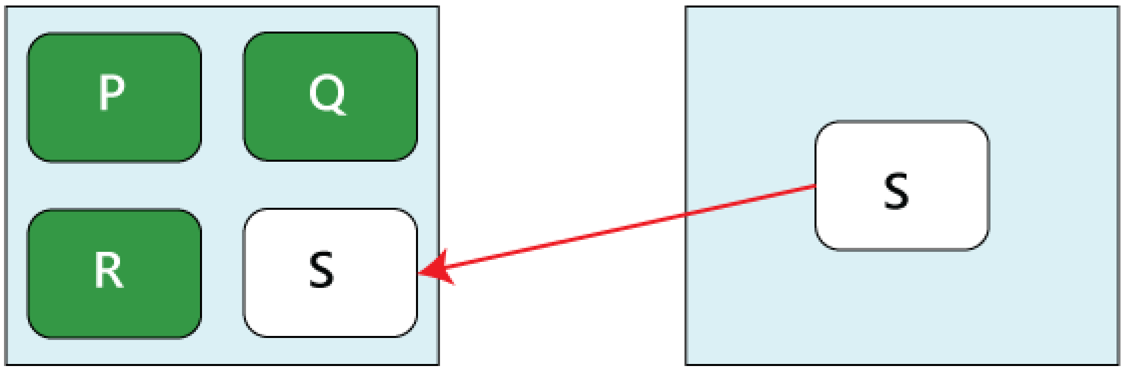
For example, Suppose we have a Gmail application to test, where **features to be tested** such as **Compose mail, Sent Items, Inbox, Drafts** and the **features which not be tested** such as **Help**, and so on which means that in the planning stage, we will decide that which functionality has to be checked or not based on the time limit given in the product.

Now **how we decide which features not to be tested?**

We have the following aspects where we can decide which feature not to be tested:

- As we see above that **Help** features is not going to be tested, as it is written and developed by the technical writer and reviewed by another professional writer.
- Let us assume that we have one application that have **P, Q, R, and S** features, which need to be developed based on the requirements. But here, the S feature has already been designed and used by some other company. So the development team will purchase S from that company and integrate with additional features such as P, Q, and R.

Now, we will not perform functional testing on the S feature because it has already been used in real-time. But we will do the integration testing, and system testing between P, Q, R, and S features because the new features might not work correctly with S feature as we can see in the below image:



- Suppose in the first release of the product, the elements that have been developed, such as **P, Q, R, 0S, T, U, V, W…..X, Y, Z**. Now the client will provide the requirements for the new features which improve the product in the second release and the new features are **A1, B2, C3, D4, and E5.**

After that, we will write the scope during the test plan as

Scope

Features to be tested

A1, B2, C3, D4, E5 (new features)

P, Q, R, S, T

Features not to be tested

W…..X, Y, Z

Therefore, we will check the new features first and then continue with the old features because that might be affected after adding the new features, which means it will also affect the impact areas, so we will do one round of regressing testing for P, Q, R…, T features.

Test methodology:

It contains information about performing a different kind of testing like Functional testing, Integration testing, and System testing, etc. on the application. In this, we will decide what type of testing; we will perform on the various features based on the application requirement. And here, we should also define that what kind of testing we will use in the testing methodologies so that everyone, like the management, the development team, and the testing team can understand easily because the testing terms are not standard.

For example, for standalone application such as **Adobe Photoshop**, we will perform the following types of testing:

Smoke testing→ Functional testing → Integration testing →System testing →Adhoc testing → Compatibility testing → Regression testing→ Globalization testing → Accessibility testing → Usability testing → Reliability testing → Recovery testing → Installation or Uninstallation testing

And suppose we have to test the https://www.jeevansathi.com/ application, so we will perform following types of testing:

Smoke testing	Functional testing	Integration testing
System testing	Adhoc testing	Compatibility test
Regression testing	Globalization testing	Accessibility testi
Usability testing	Performance testing	

Approach

This attribute is used to describe the flow of the application while performing testing and for the future reference.

We can understand the flow of the application with the help of below aspects:

- **By writing the high-level scenarios**

- **By writing the flow graph**

By writing the high-level scenarios

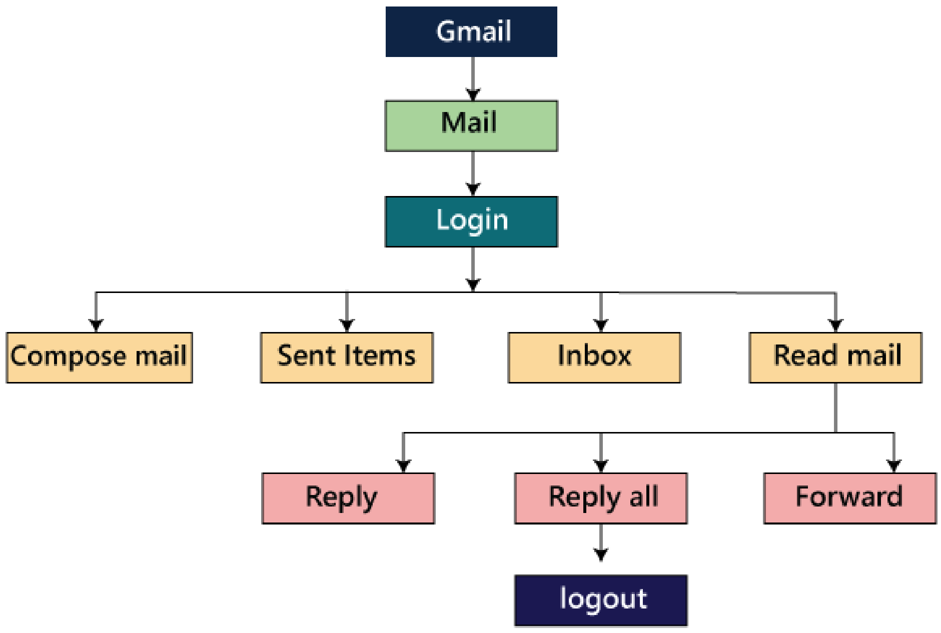
For example, suppose we are testing the **Gmail** application:

- Login to Gmail- sends an email and check whether it is in the Sent Items page
- Login to …….
- ……
- …....

We are writing this to describe the approaches which have to be taken for testing the product and only for the critical features where we will write the high-level scenarios. Here, we will not be focusing on covering all the scenarios because it can be decided by the particular test engineer that which features have to be tested or not.

By writing the flow graph

The flow graph is written because writing the high-level scenarios are bit time taking process, as we can see in the below image:



We are creating flow graphs to make the following benefits such as:

- Coverage is easy
- Merging is easy

The approach can be classified into two parts which are as following:

- Top to bottom approach
- Bottom to top approach

Assumption

It contains information about a problem or issue which maybe occurred during the testing process and when we are writing the test plans, the assured assumptions would be made like resources and technologies, etc.

Risk

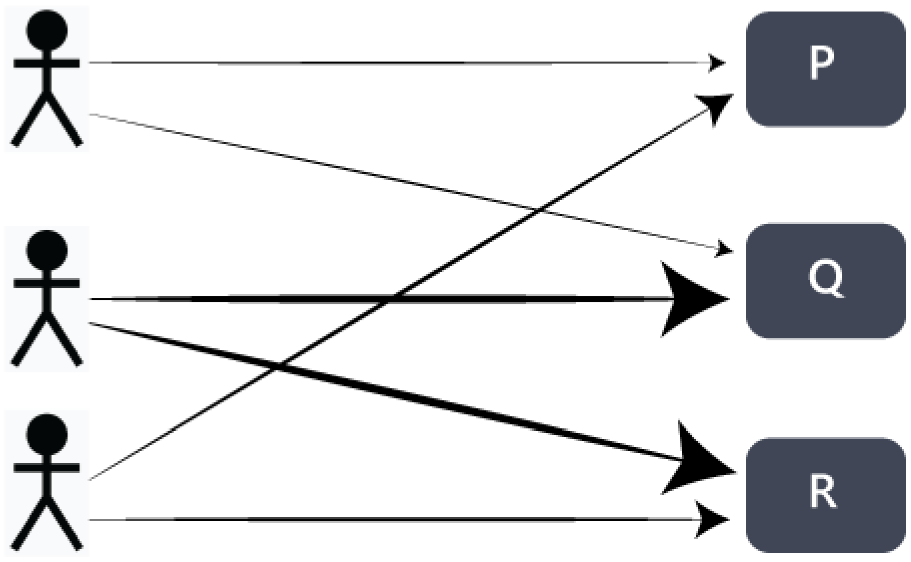
These are the challenges which we need to face to test the application in the current release and if the assumptions will fail then the risks are involved.

For example, the effect for an application, release date becomes postponed.

Mitigation Plan or Contingency Plan

It is a back-up plan which is prepared to overcome the risks or issues.

Let us see one example for assumption, risk, and the contingency plan together because they are co-related to each other.



In any product, the **assumption** we will make is that the all 3 test engineers will be there until the completion of the product and each of them is assigned different modules such as P, Q, and R. In this particular scenario, the **risk** could be that if the test engineer left the project in the middle of it.

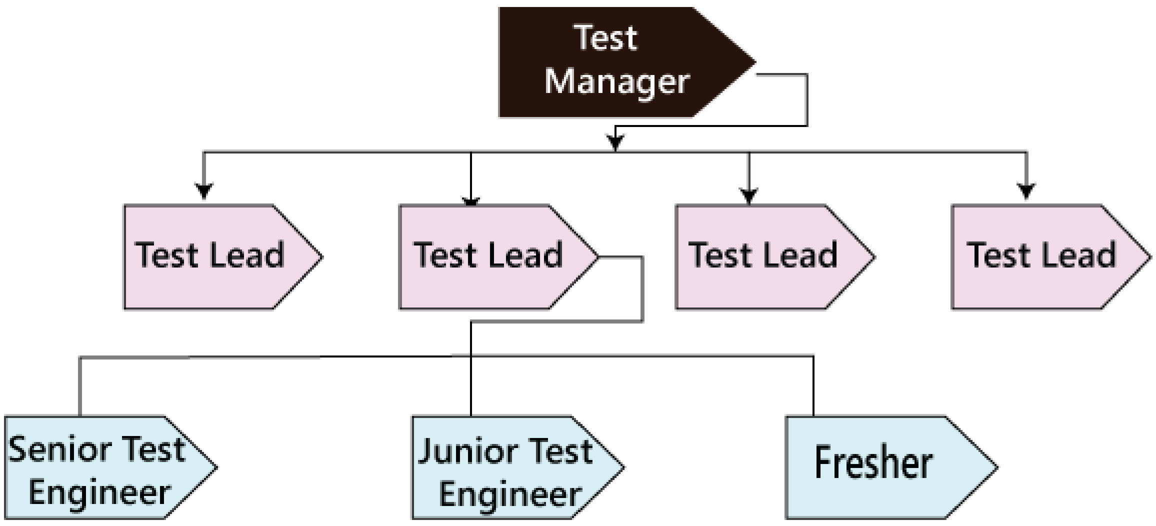
Therefore, the **contingency plan** will be assigned a primary and subordinate owner to each feature. So if the one test engineer will leave, the subordinate owner takes over that specific feature and also helps the new test engineer, so he/she can understand their assigned modules.

The assumptions, risk, and mitigation or contingency plan are always precise on the product itself. The various types of risks are as follows:

- Customer perspective
- Resource approach
- Technical opinion

Role & Responsibility

It defines the complete task which needs to be performed by the entire testing team. When a large project comes, then the **Test Manager** is a person who writes the test plan. If there are 3-4 small projects, then the test manager will assign each project to each Test Lead. And then, the test lead writes the test plan for the project, which he/she is assigned.



Let see one example where we will understand the roles and responsibility of the Test manager, test lead, and the test engineers.

Role: Test Manager

Name: Ryan

Responsibility:

- Prepare(write and review) the test plan
- Conduct the meeting with the development team
- Conduct the meeting with the testing team
- Conduct the meeting with the customer
- Conduct one monthly stand up meeting
- Sign off release note
- Handling Escalations and issues

Role: Test Lead

Name: Harvey

Responsibility:

- Prepare(write and review) the test plan
- Conduct daily stand up meeting
- Review and approve the test case
- Prepare the RTM and Reports
- Assign modules
- Handling schedule

Role: Test Engineer 1, Test Engineer 2 and Test Engineer 3

Name: Louis, Jessica, Donna

Assign modules: M1, M2, and M3

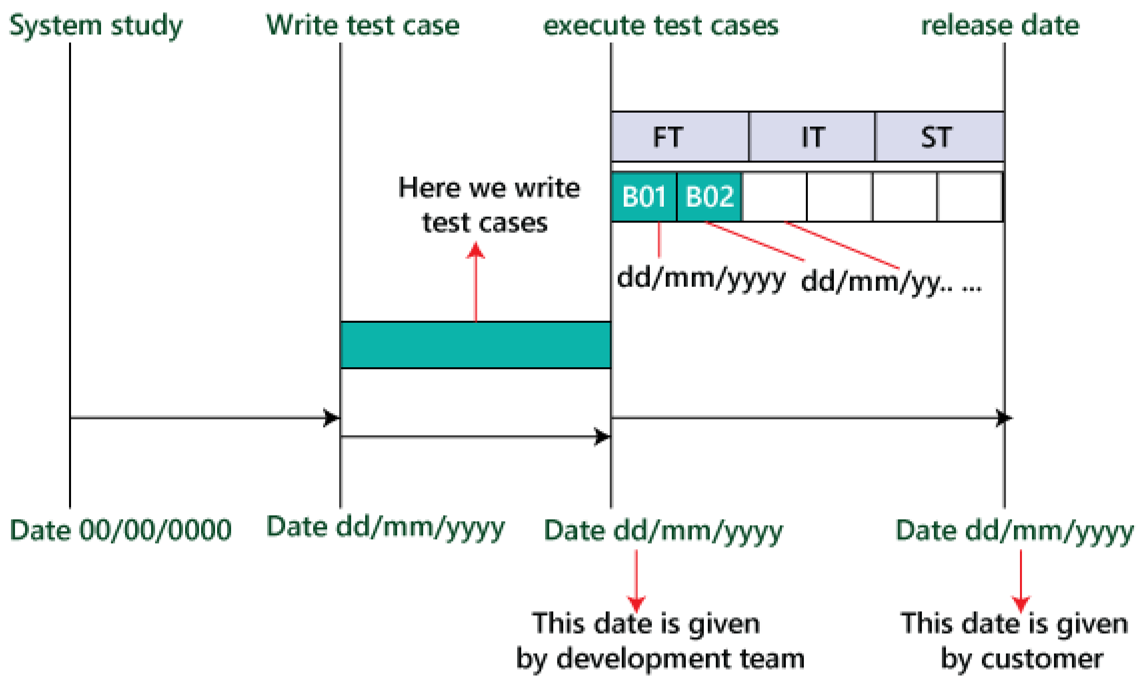
Responsibility:

- Write, Review, and Execute the test documents which consists of test case and test scenarios
- Read, review, understand and analysis the requirement
- Write the flow of the application
- Execute the test case

- RTM for respective modules
- Defect tracking
- Prepare the test execution report and communicate it to the Test Lead.

Schedule

It is used to explain the timing to work, which needs to be done or this attribute covers when exactly each testing activity should start and end? And the exact data is also mentioned for every testing activity for the particular date.



Therefore as we can see in the below image that for the particular activity, there will be a starting date and ending date; for each testing to a specific build, there will be the specified date.

For example

- Writing test cases
- Execution process

Defect tracking

It is generally done with the help of tools because we cannot track the status of each bug manually. And we also comment about how we communicate the bugs which are identified during the testing process and send it back to the development team and how the development team will reply. Here we also mention the priority of the bugs such as high, medium, and low.

Following are various aspects of the defect tracking:

- **Techniques to track the bug**
  …..
  ……
  ……
  ……
- **Bug tracking tools**
  We can comment on the name of the tool, which we will use for tracking the bugs. Some of the most commonly used bug tracking tools are Jira, Bugzilla, Mantis, and Trac, etc.<
- **Severity**
  The severity could be as following:
  Blocker or showstopper
  …..
  ….. (Explain it with an example in the test plan)
  For example, there will be a defect in the module; we cannot go further to test other modules because if the bug is blocked, we can proceed to check other modules.
  Critical
  ……
  ….. (Explain it with an example in the test plan)
  In this situation, the defects will affect the business.
  Major
  ….
  …. (Explain it with an example in the test plan)
  Minor
  …..
  ….. (Explain it with an example in the test plan)
  These defects are those, which affect the look and feel of the application.
- **Priority**
  High-P1
  …..

Medium-P2

…..

Low-P3

…..

…..

P4

Therefore, based on the priority of bugs liike high, medium, and low, we will categorize it as P1, P2, P3, and P4.

Test Environments

These are the environments where we will test the application, and here we have two types of environments, which are of **software** and **hardware** configuration.

The **software configuration** means the details about different **Operating Systems** such as **Windows, Linux, UNIX, and Mac** and various **Browsers** like **Google Chrome, Firefox, Opera, Internet Explorer**, and so on.

And the **hardware configuration** means the information about different sizes of **RAM, ROM, and the Processors**.

For example

- The **Software** includes the following:

Server

Operating system	Linux
Webserver	Apache Tomcat
Application server	Websphere
Database server	Oracle or MS-SQL Server

Note: The above servers are the serves that are used by the testing team to test the application.

Client

Operating System	Window XP, Vista 7
Browsers	Mozilla Firefox, Google Chrome, Internet Explorer, Internet Explor Explorer 8

Note: The above details provide the various operating systems and browsers in which the testing team will test the application.

- The **Hardware** includes the following:

Server: Sun StarCat 1500

This particular server can be used by the testing team to test their application.

Client:

It has the following configuration, which is as follows:

Processor	Intal2GHz
RAM	2GB
….	….

Note: It will provide the configuration of the systems of the test engineers that is the testing team.

- **Process to install the software**
  ……
  …..
  …..

The development team will provide the configuration of how to install the software. If the development team will not yet provide the process, then we will write it as Task-Based Development (TBD) in the test plan.

Entry and Exit criteria

It is a necessary condition, which needs to be satisfied before starting and stopping the testing process.

Entry Criteria

The entry criteria contain the following conditions:

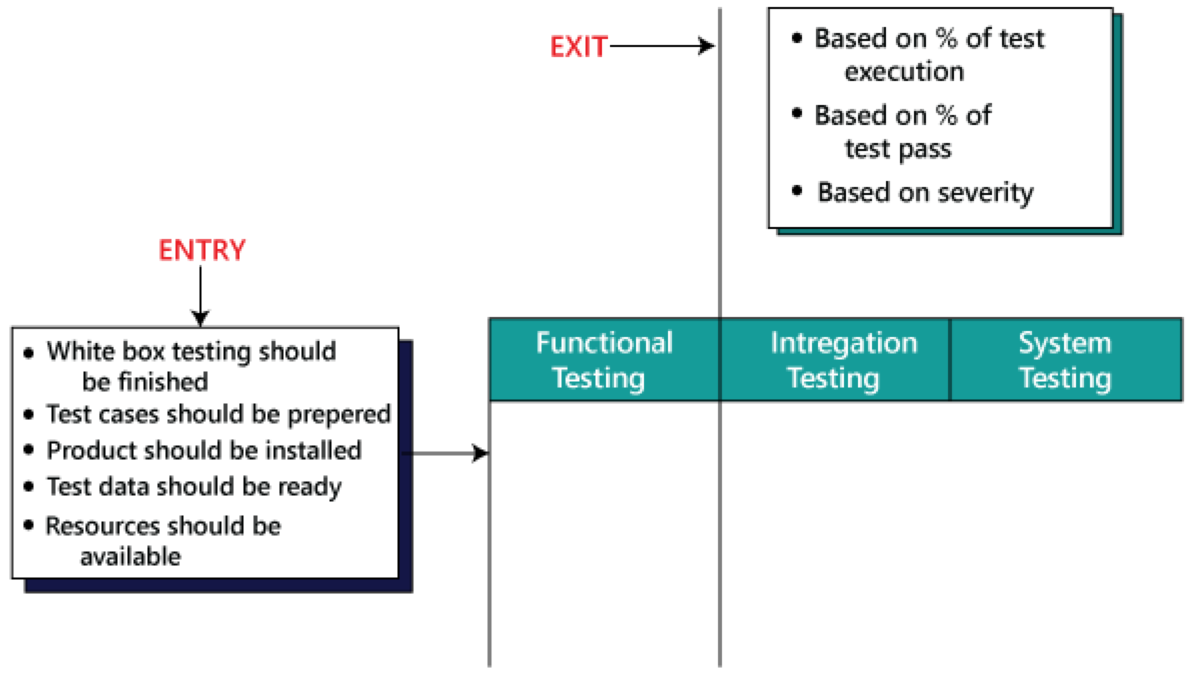
- White box testing should be finished.
- Understand and analyze the requirement and prepare the test documents or when the test documents are ready.
- Test data should be ready.
- Build or the application must be prepared
- Modules or features need to be assigned to the different test engineers.
- The necessary resource must be ready.

Exit Criteria

The exit criteria contain the following conditions:

- When all the test cases are executed.
- Most of the test cases must be **passed**.
- Depends on severity of the bugs which means that there must not be any blocker or major bug, whereas some minor bugs exist.

Before we start performing functional testing, all the above **Entry Criteria** should be followed. After we performed functional testing and before we will do integration testing, then the **Exit criteria of** the functional testing should be followed because the % of exit criteria are decided by the meeting with both development and test manager because their collaboration can achieve the percentage. But if the exit criteria of functional testing are not followed, then we cannot proceed further to integration testing.



EXIT
• Based on % of test execution
• Based on % of test pass
• Based on severity
ENTRY
• White box testing should be finished
• Test cases should be prepered
• Product should be installed
• Test data should be ready
• Resources should be available
Functional Testing
Intregation Testing
System Testing

Here **based on the severity** of the bug's means that the testing team would have decided that to proceed further for the next phases.

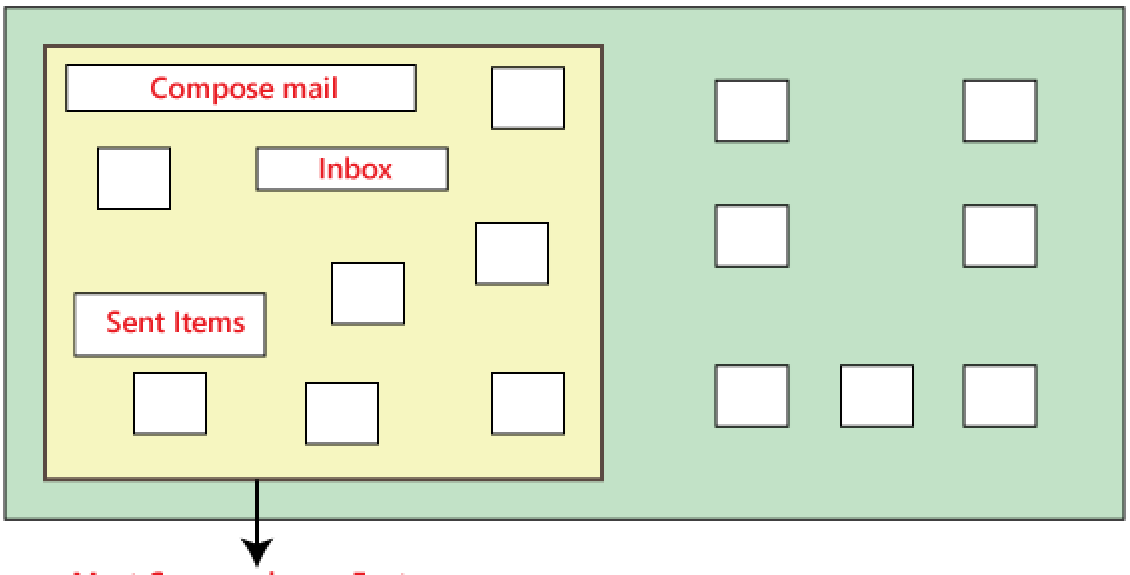
Test Automation

In this, we will decide the following:

- Which feature has to be automated and not to be automated?
- Which test automation tool we are going to use on which automation framework?

We automate the test case only after the first release.

Here the question arises that on what basis **we** will **decide which features have to be tested?**



Most Commonly use Features

In the above image, as we can see that the most commonly used features need to test again and again. Suppose we have to check the Gmail application where the essential features are **Compose mail, Sent Items, and Inbox**. So we will test these features because while performing manual testing, it takes more time, and it also becomes a monotonous job.

Now, **how we decide which features are not going to be tested?**

Suppose **the Help** feature of the Gmail application is not tested again and again because these features are not regularly used, so we do not need to check it frequently.

But **if some features are unstable and have lots of bugs**, which means that we will not test those features because it has to be tested again and again while doing manual testing.

If **there is a feature that has to be tested frequently**, but we are expecting the requirement change for that feature, so we do not check it because changing the manual test cases is more comfortable as compared to change in the automation test script.

Effort estimation

In this, we will plan the effort need to be applied by every team member.

Test Deliverable

These are the documents which are the output from the testing team, which we handed over to the customer along with the product. It includes the following:

- **Test plan**
- **Test Cases**
- **Test Scripts**
- **RTM(Requirement Traceability Matrix)**
- **Defect Report**
- **Test Execution Report**
- **Graphs and metrics**
- **Release Notes**

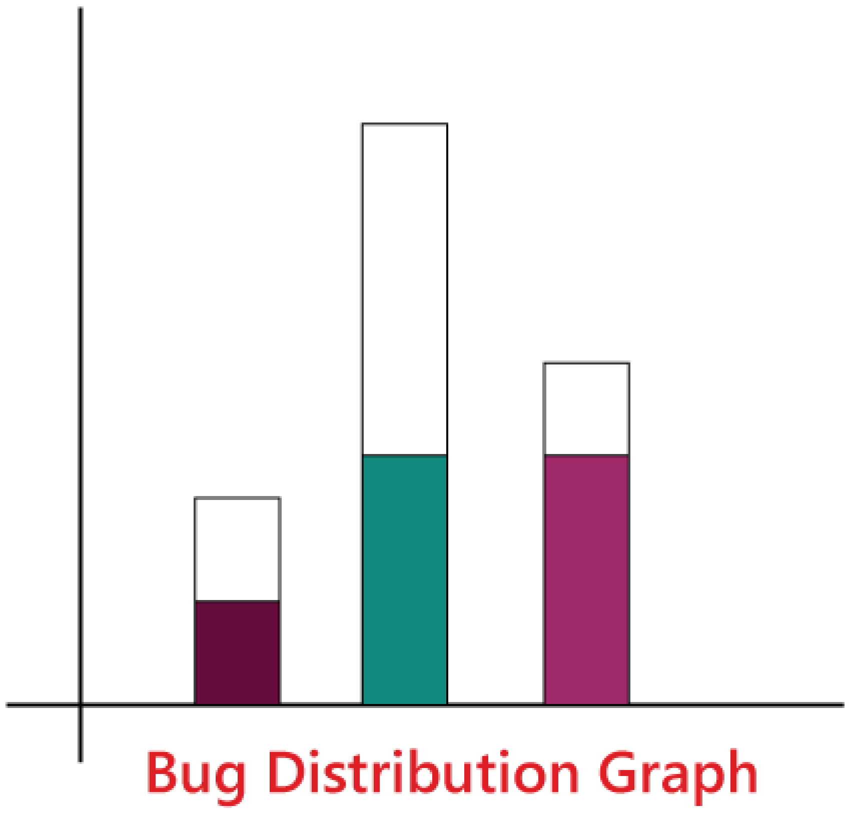
Graphs and Metrics

Graph

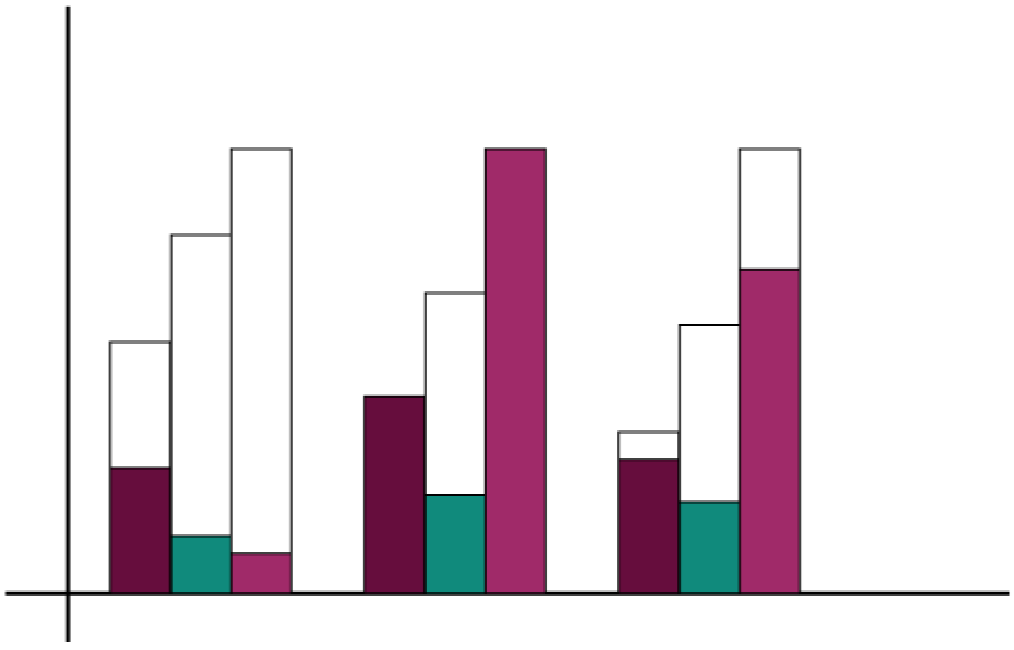
In this, we will discuss about the types of **graphs** we will send, and we will also provide a sample of each graph.

As we can see, we have five different graphs that show the various aspects of the testing process.

Graph1: In this, we will show how many defects have been identified and how many defects have been fixed in every module.

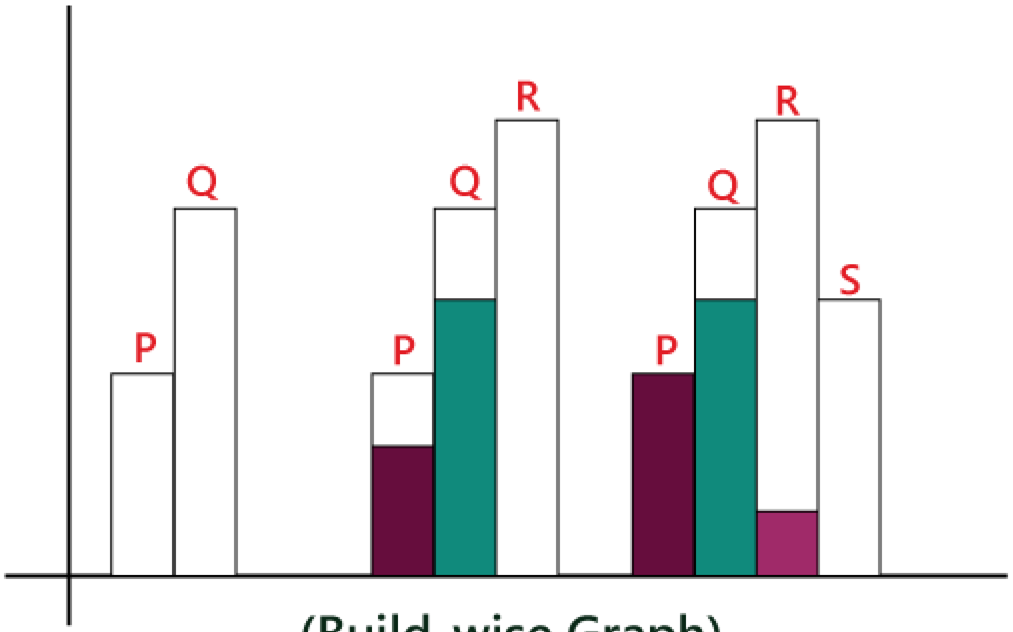


Graph 2: Figure one shows how many critical, major, and minor defects have been identified for every module and how many have been fixed for their respective modules.



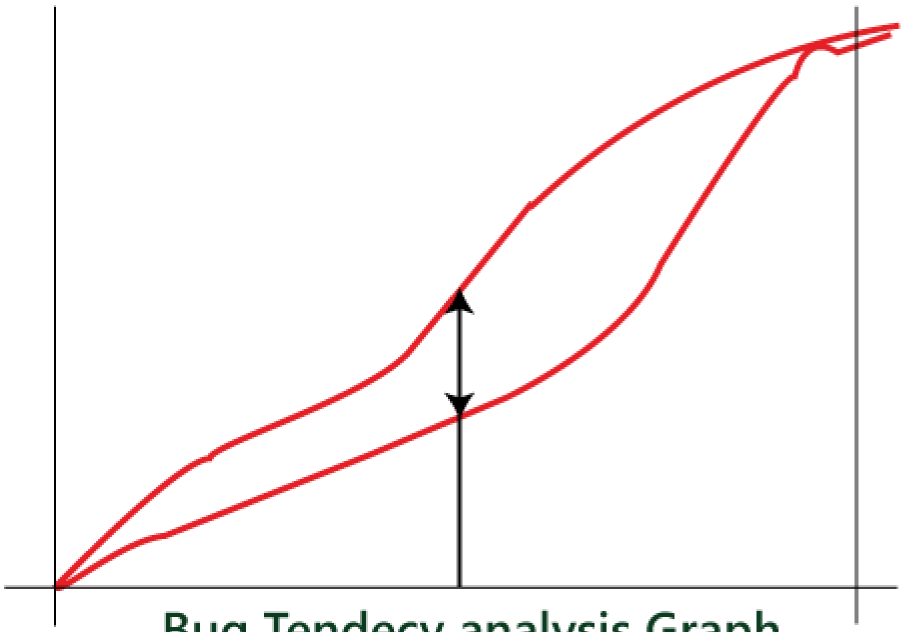
Graph3: In this particular graph, we represent the **build wise graph**, which means that in every builds how many defects have been identified and fixed for

every module. Based on the module, we have determined the bugs. We will add **R** to show the number of defects in P and Q, and we also add **S** to show the defects in P, Q, and R.



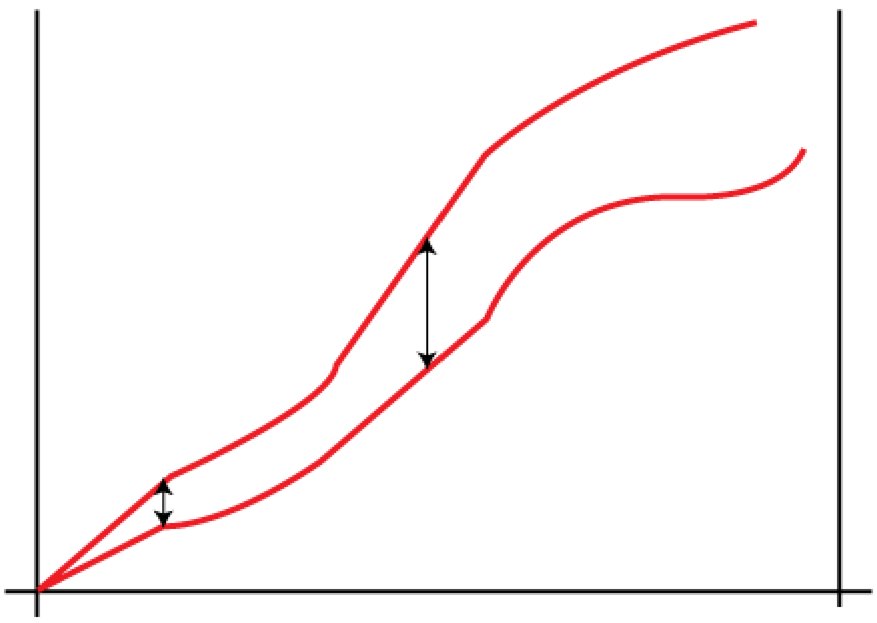
(Build-wise Graph)

Graph4: The test lead will design the **Bug Trend analysis** graph which is created every month and send it to the Management as well. And it is just like prediction which is done at the end of the product. And here, we can also **rate the bug fixes** as we can observe that **arc** has an **upward tendency** in the below image.

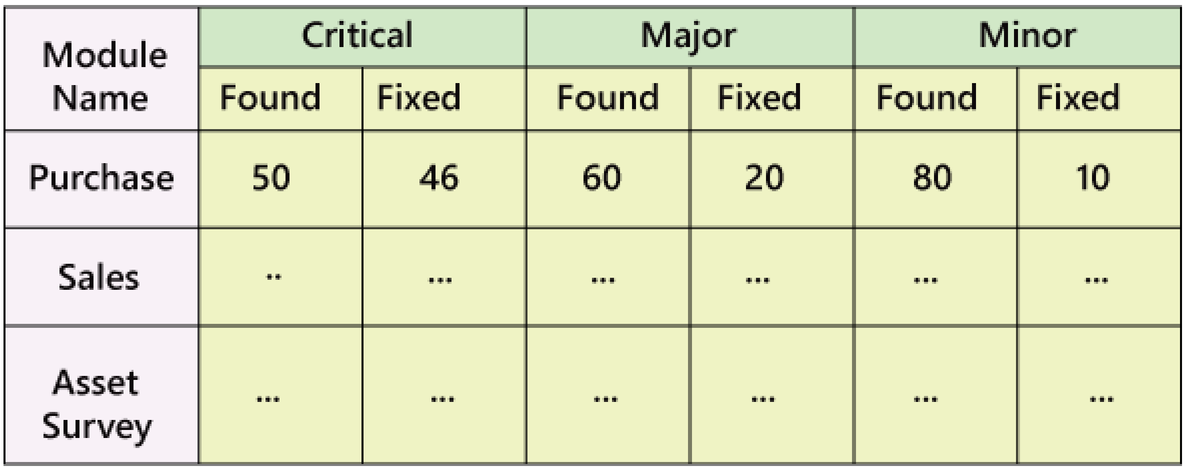


Bug Tendecy analysis Graph

Graph5: The **Test Manager** has designed this type of graph. This graph is intended to understand the gap in the assessment of bugs and the actual bugs which have been occurred, and this graph also helps to improve the evaluation of bugs in the future.



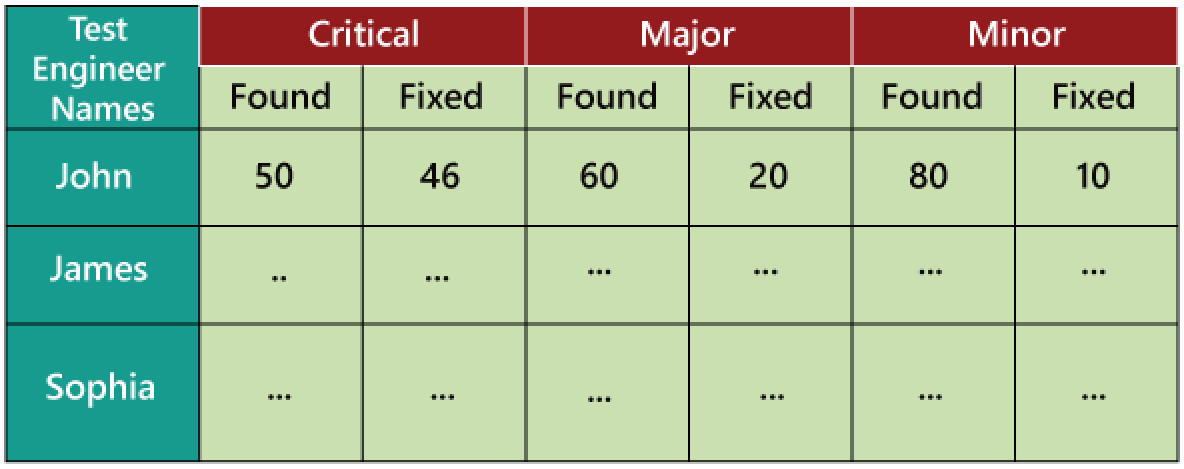
Metrics



Module Name	Critical		Major		Minor	
	Found	Fixed	Found	Fixed	Found	Fixed
Purchase	50	46	60	20	80	10
Sales	..	...	...	...	...	...
Asset Survey	...	...	...	...	...	...

As above, we create the bug distribution graph, which is in the figure 1, and with the help of above mention data, we will design the metrics as well.

For example

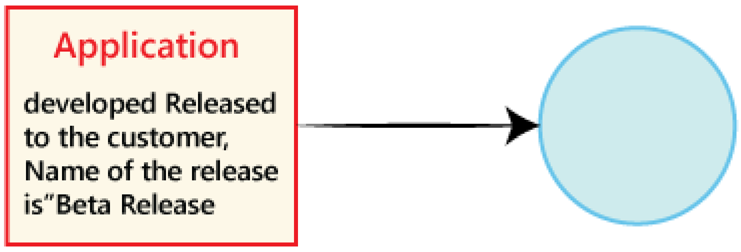


Test Engineer Names	Critical		Major		Minor	
	Found	Fixed	Found	Fixed	Found	Fixed
John	50	46	60	20	80	10
James	..	...	...	...	...	...
Sophia	...	...	...	...	...	...

In the above figure, we retain the records of all the test engineers in a particular project and how many defects have been identified and fixed. We can also use this data for future analysis. When a new requirement comes, we can decide whom to provide the challenging feature for testing based on the number of defects they have found earlier according to the above metrics. And we will be in a better situation to know who can handle the problematic features very well and find maximum numbers of defects.

Release Note: It is a document that is prepared during the release of the product and signed by the Test Manager.

In the below image, we can see that the final product is developed and deployed to the customer, and the latest release name is **Beta**.


Application
developed Released to the customer, Name of the release is"Beta Release

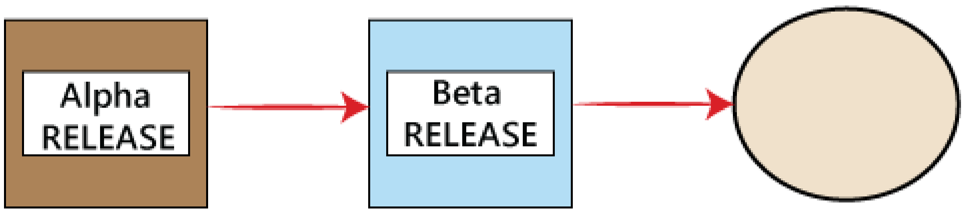
The **Release note** consists of the following:

- User manual.
- List of pending and open defects.
- List of added, modified, and deleted features.
- List of the platform (Operating System, Hardware, Browsers) on which the product is tested.

- Platform in which the product is not tested.
- List of bugs fixed in the current release, and the list of fixed bugs in the previous release.
- Installation process
- Versions of the software

For Example

Suppose that **Beta** is the second release of the application after the first release **Alpha** is released. Some of the defect identified in the first released and that has been fixed in the later released. And here, we will also point out the list of newly added, modified, and deleted features from alpha release to the beta release.

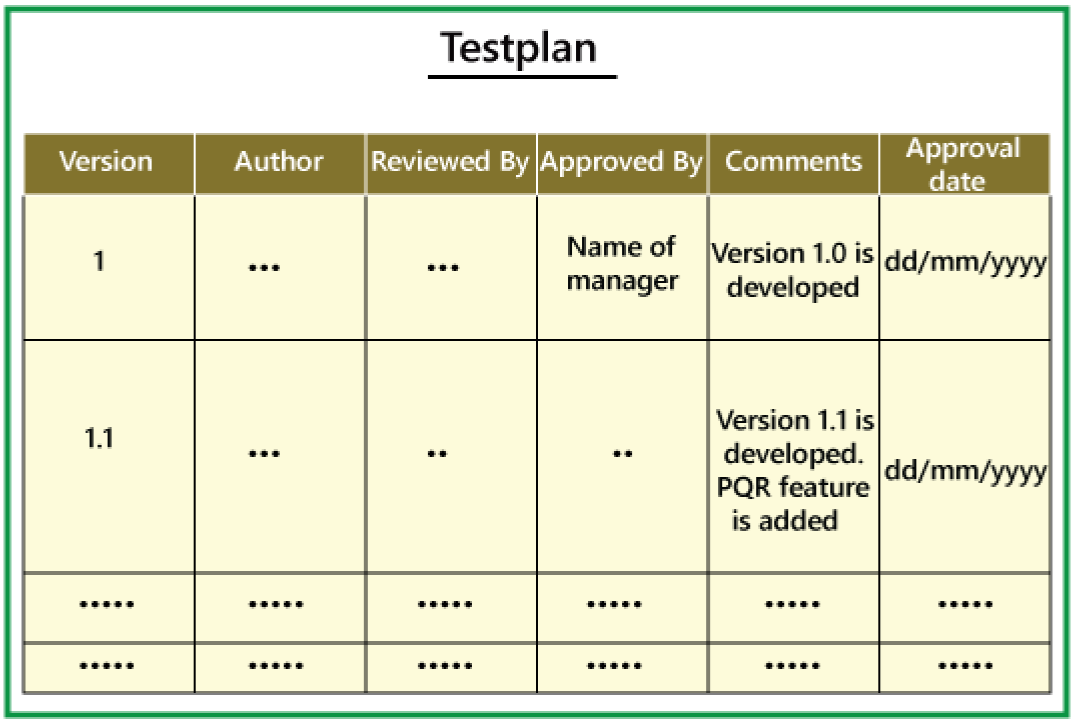


Template

This part contains all the templates for the documents that will be used in the product, and all the test engineers will use only these templates in the project to maintain the consistency of the product. Here, we have different types of the template which are used during the entire testing process such as:

- Test case template
- Test case review template
- RTM Template
- Bug Report Template
- Test execution Report

Let us see one sample of test plan document



Testplan

Version	Author	Reviewed By	Approved By	Comments	Approval date
1	•••	•••	Name of manager	Version 1.0 is developed	dd/mm/yyyy
1.1	•••	••	••	Version 1.1 is developed. PQR feature is added	dd/mm/yyyy
•••••	•••••	•••••	•••••	•••••	•••••
•••••	•••••	•••••	•••••	•••••	•••••

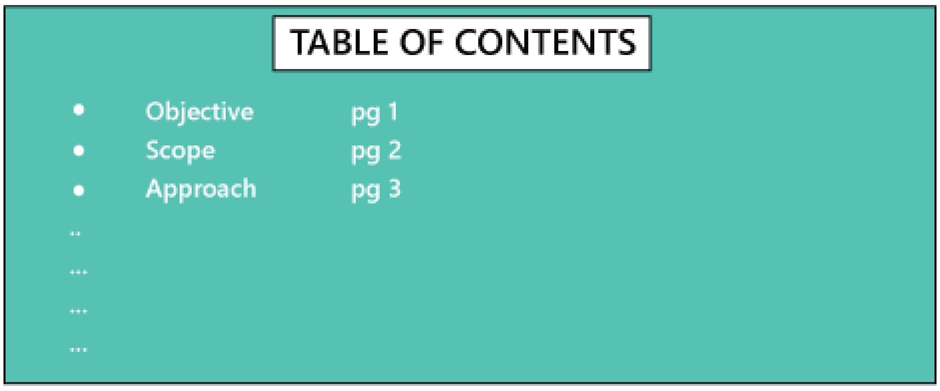
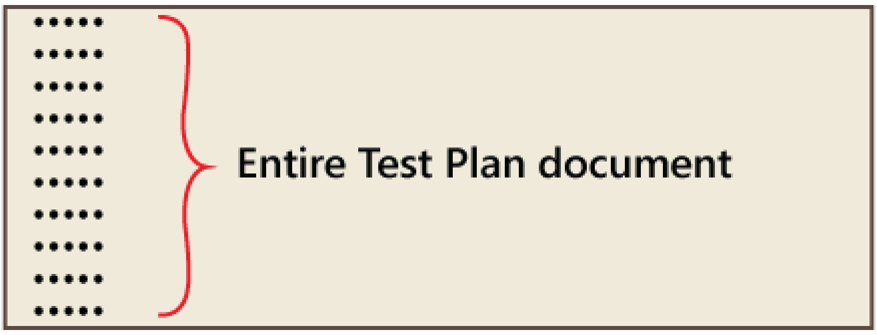
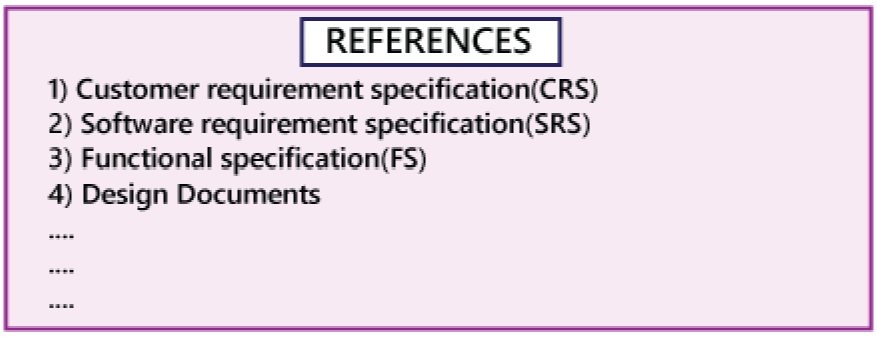


TABLE OF CONTENTS

• Objective	pg 1
• Scope	pg 2
• Approach	pg 3
..	
...	
...	
...	




REFERENCES

1) Customer requirement specification(CRS)
2) Software requirement specification(SRS)
3) Functional specification(FS)
4) Design Documents

....
....
....

In-Page 1, we primarily fill only the **Versions, Author, Comments, and Reviewed By** fields, and after the manager approves it, we will mention the details in the **Approved By and Approval Date** fields.

Mostly the test plan is approved by the Test Manager, and the test engineers only reviews it. And when the new features come, we will modify the test plan and do the necessary modification in **Version** field, and then it will be sent again for further review, update, and approval of the manager. The test plan must be updated whenever any changes have occurred. On page 20, the **References** specify the details about all the documents which we are going to use to write the test plan document.

Note:

Who writes the test plan?

- Test Lead→60%
- Test Manager→20%
- Test Engineer→20%

Therefore, as we can see from above that in 60% of the product, the test plan is written by the Test Lead.

Who reviews the Test Plan?

- Test Lead
- Test Manager
- Test engineer
- Customer
- Development team

The Test Engineer review the Test plan for their module perspective and the test manager review the Test plan based on the customer opinion.

Who approve the test Plan?

- Customer
- Test Manager

Who writes the test case?

- Test Lead
- Test Engineer

Who review the test case?

- Test Engineer
- Test Lead
- Customer
- Development Team

Who approves the Test cases?

- Test Manager
- Test Lead
- Customer

Test Plan Guidelines

- Collapse your test plan.
- Avoid overlapping and redundancy.
- If you think that you do not need a section that is already mentioned above, then delete that section and proceed ahead.
- Be specific. For example, when you specify a software system as the part of the test environment, then mention the software version instead of only name.
- Avoid lengthy paragraphs.
- Use lists and tables wherever possible.
- Update plan when needed.

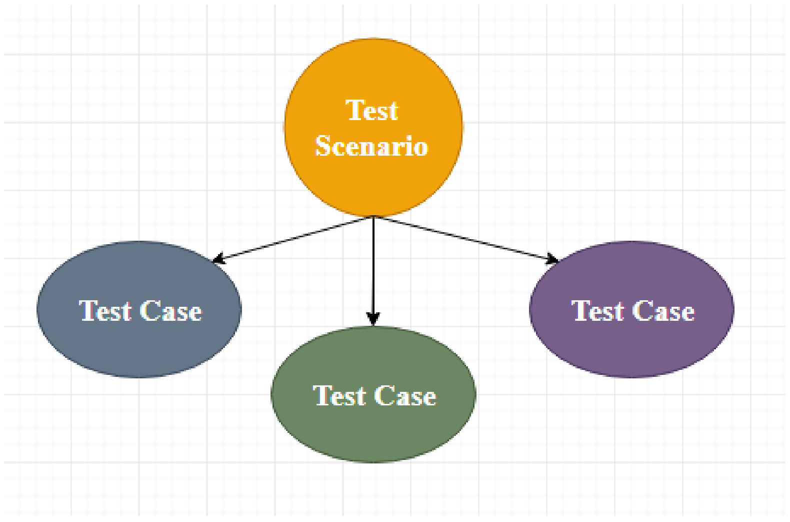
- Do not use an outdated and unused document.

Importance of Test Plan

- The test plan gives direction to our thinking. This is like a rule book, which must be followed.
- The test plan helps in determining the necessary efforts to validate the quality of the software application under the test.
- The test plan helps those people to understand the test details that are related to the outside like developers, business managers, customers, etc.
- Important aspects like test schedule, test strategy, test scope etc are documented in the test plan so that the management team can review them and reuse them for other similar projects.

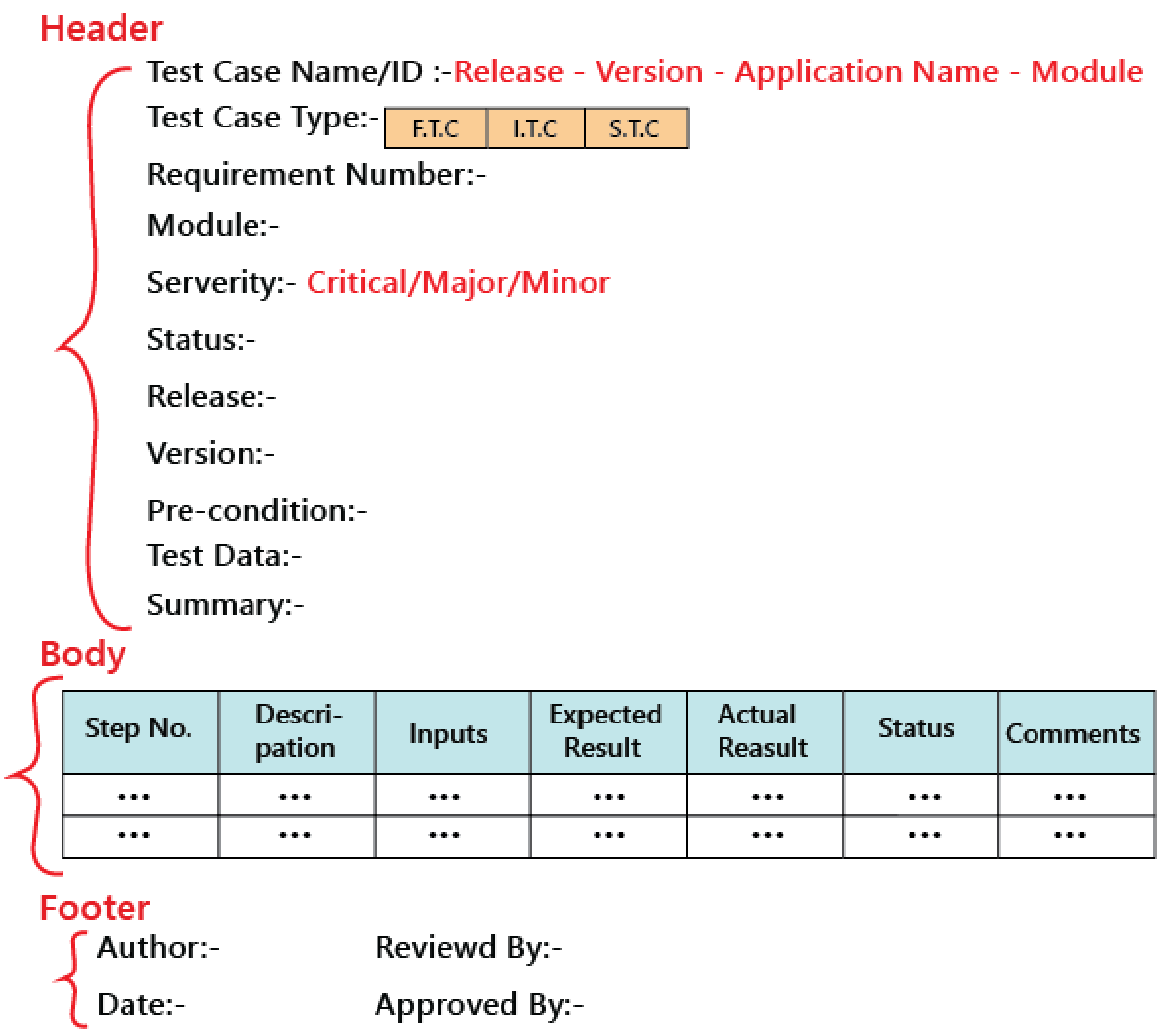
Test case:

- The test case is defined as a group of conditions under which a tester determines whether a software application is working as per the customer's requirements or not. Test case designing includes preconditions, case name, input conditions, and expected result. A test case is a first level action and derived from test scenarios.



Test case template

The primary purpose of writing a test case is to achieve the efficiency of the application.



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:-

As we know, the **actual result** is written after the test case execution, and most of the time, it would be same as the **expected result**. But if the test step will fail, it will be different. So, the actual result field can be skipped, and in **the Comments** section, we can write about the bugs.

And also, the **Input field** can be removed, and this information can be added to the **Description field**.

The above template we discuss above is not the standard one because it can be different for each company and also with each application, which is based on the test engineer and the test lead. But, for testing one application, all the test engineers should follow a usual template, which is formulated.

The test case should be written in simple language so that a new test engineer can also understand and execute the same.

In the above sample template, the header contains the following:

Step number

It is also essential because if step number 20 is failing, we can document the bug report and hence prioritize working and also decide if it’s a critical bug.

Test case type

It can be functional, integration or system test cases or positive or negative or positive and negative test cases.

Release

One release can contain many versions of the release.

Pre-condition

These are the necessary conditions that need to be satisfied by every test engineer before starting the test execution process. Or it is the data configuration or the data setup that needs to be created for the testing.

For example: In an application, we are writing test cases to add users, edit users, and delete users. The per-condition will be seen if user A is added before editing it and removing it.

Test data

These are the values or the input we need to create as per the per-condition.

For example, Username, Password, and account number of the users.

The test lead may be given the test data like username or password to test the application, or the test engineer may themself generate the username and password.

Severity

The severity can be **major, minor, and critical**, the severity in the test case talks about the importance of that particular test cases. All the text execution process always depends on the severity of the test cases.

We can choose the severity based on the module. There are many features include in a module, even if one element is critical, we claim that test case to be critical. It depends on the functions for which we are writing the test case.

For example, we will take the Gmail application and let us see the severity based on the modules:

Modules	Severity
Login	Critical
Help	Minor
Compose mail	Critical
Setting	Minor
Inbox	Critical
Sent items	Major
Logout	Critical

And for the banking application, the severity could be as follows:

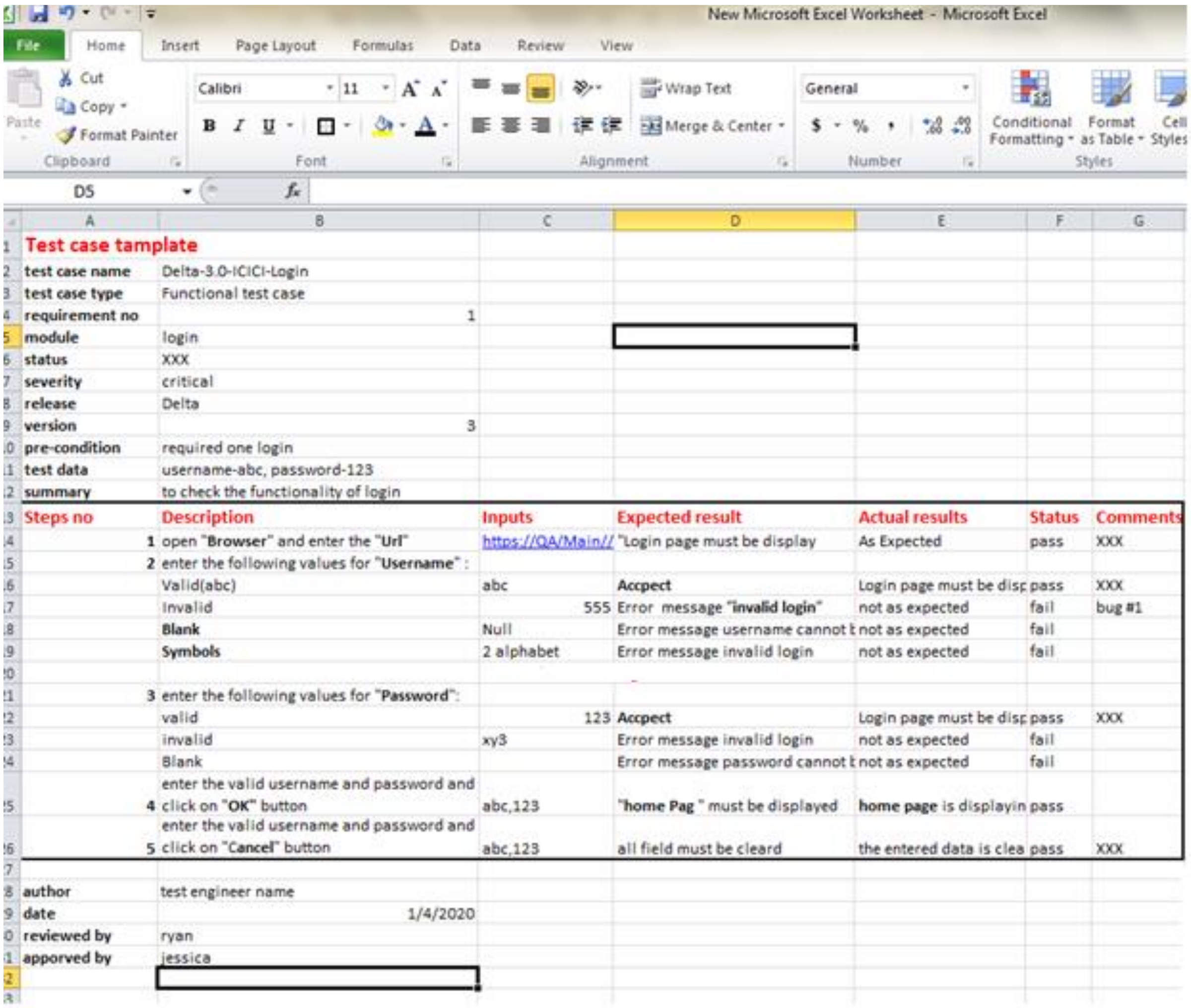
Modules	Severity
Amount transfer	Critical
Feedback	Minor

Brief description

The test engineer has written a test case for a particular feature. If he/she comes and reads the test cases for the moment, he/she will not know for what feature has written it. So, the brief description will help them in which feature test case is written.

Example of a test case template

Here, we are writing a test case for the **ICICI application's Login** module:



A	B	C	D	E	F	G
Test case tamplate						
test case name	Delta-3.0-ICICI-Login					
test case type	Functional test case					
requirement no	1					
module	login					
status	XXX					
severity	critical					
release	Delta					
version	3					
pre-condition	required one login					
test data	username-abc, password-123					
summary	to check the functionality of login					
Steps no	**Description**	**Inputs**	**Expected result**	**Actual results**	**Status**	**Comments**
1	open "**Browser**" and enter the "**Url**"	https://QA/Main//	"Login page must be display	As Expected	pass	XXX
2	enter the following values for "**Username**" :					
	Valid(abc)	abc	**Accpect**	Login page must be disp	pass	XXX
	Invalid	555	Error message "**invalid login**"	not as expected	fail	bug #1
	Blank	Null	Error message username cannot b	not as expected	fail	
	Symbols	2 alphabet	Error message invalid login	not as expected	fail	
3	enter the following values for "**Password**":					
	valid	123	**Accpect**	Login page must be disp	pass	XXX
	invalid	xy3	Error message invalid login	not as expected	fail	
	Blank		Error message password cannot b	not as expected	fail	
4	enter the valid username and password and click on "**OK**" button	abc,123	"**home Pag** " must be displayed	**home page** is displayin	pass	
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
author	test engineer name					
date	1/4/2020					
reviewed by	ryan					
apporved by	jessica					

Types of test cases

We have a different kind of test cases, which are as follows:

- **Function test cases**
- **Integration test cases**
- **System test cases**

The functional test cases

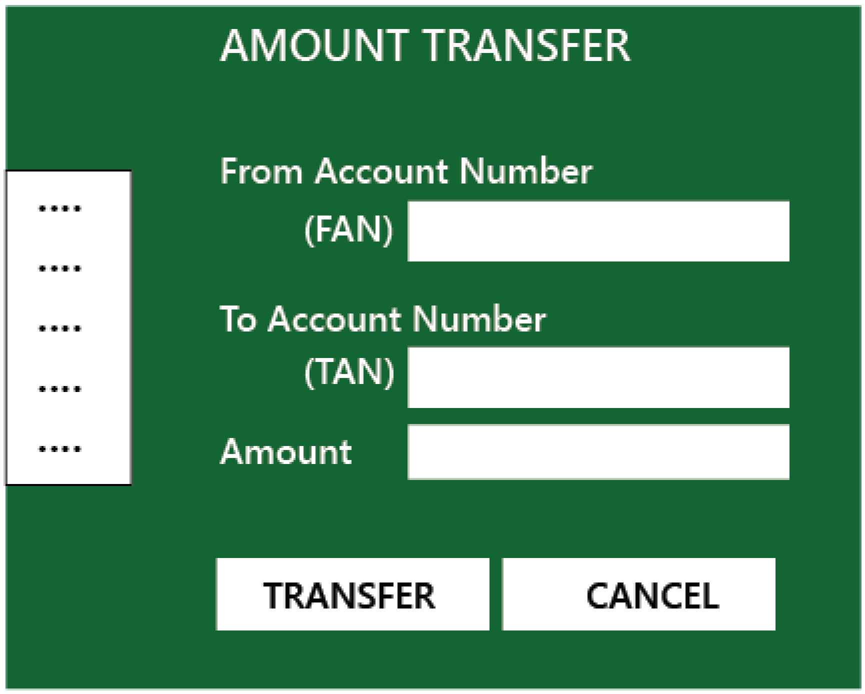
Firstly, we check for which field we will write test cases and then describe accordingly.

In functional testing or if the application is data-driven, we require the input column else; it is a bit time-consuming.

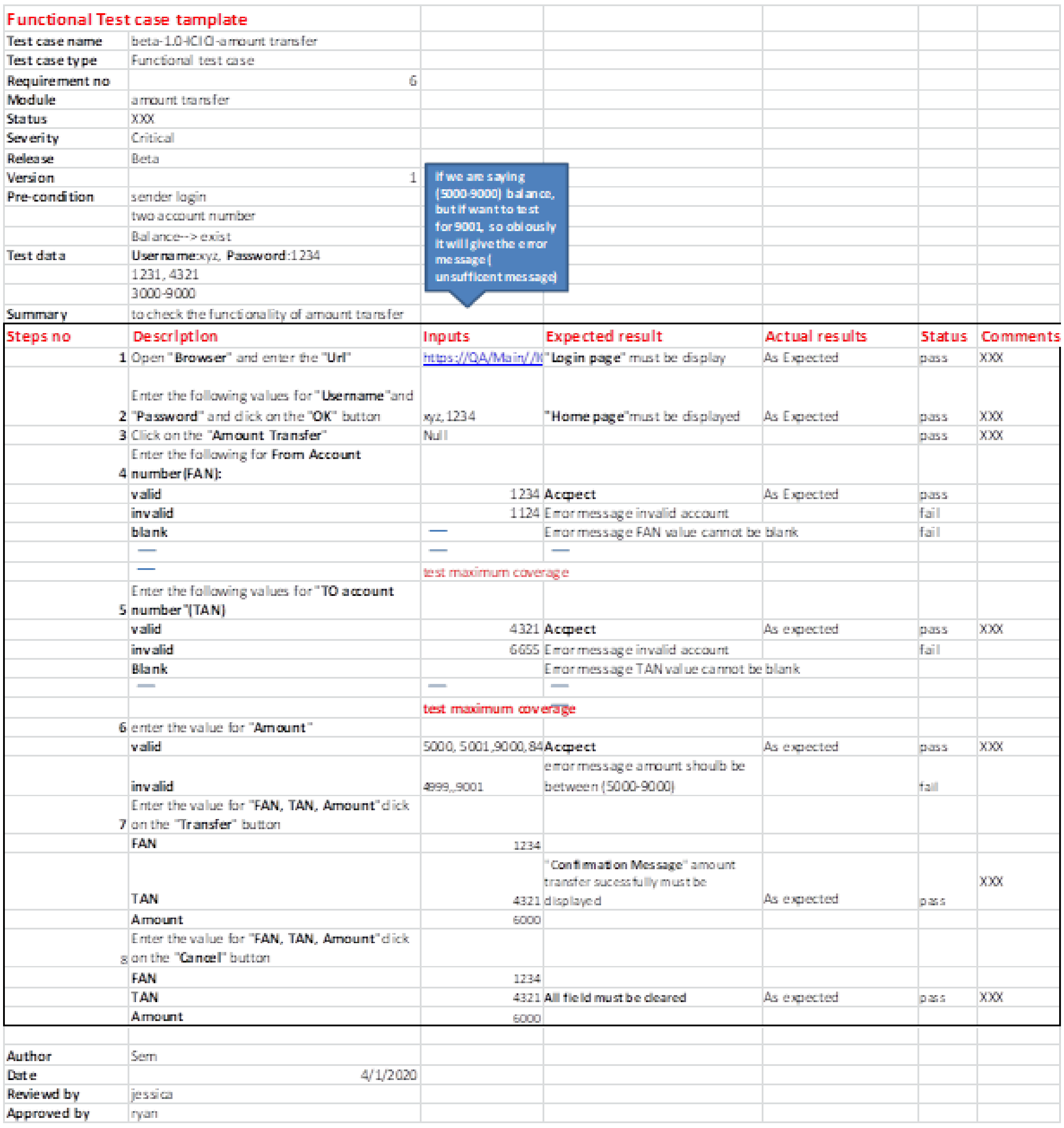
Rules to write functional test cases:

- In the expected results column, try to use **should be** or **must be**.
- Highlight the Object names.
- We have to describe only those steps which we required the most; otherwise, we do not need to define all the steps.
- To reduce the excess execution time, we will write steps correctly.
- Write a generic test case; do not try to hard code it.

Let say it is the amount transfer module, so we are writing the functional test cases for it and then also specifies that it is not a login feature.



The functional test case for amount transfer module is in the below Excel file:



Functional Test case tamplate						
Test case name	beta-1.0-ICICI-amount transfer					
Test case type	Functional test case					
Requirement no	6					
Module	amount transfer					
Status	XXX					
Severity	Critical					
Release	Beta					
Version	1					
Pre-condition	sender login					
	two account number					
	Balance--> exist					
Test data	**Username**:xyz, **Password**:1234					
	1231, 4321					
	3000-9000					
Summary	to check the functionality of amount transfer					
Steps no	**Description**	**Inputs**	**Expected result**	**Actual results**	**Status**	**Comments**
1	Open "**Browser**" and enter the "**Url**"	https://QA/Main/ /K	"**Login page**" must be display	As Expected	pass	XXX
2	Enter the following values for "**Username**" and "**Password**" and click on the "**OK**" button	xyz, 1234	"**Home page**" must be displayed	As Expected	pass	XXX
3	Click on the "**Amount Transfer**"	Null			pass	XXX
4	Enter the following for **From Account number (FAN):**					
	valid	1234	**Accpect**	As Expected	pass	
	invalid	1124	Error message invalid account		fail	
	blank	—	Error message FAN value cannot be blank		fail	
	—	—	—			
	—	test maximum coverage				
5	Enter the following values for "**TO account number**"(**TAN**)					
	valid	4321	**Accpect**	As expected	pass	XXX
	invalid	6655	Error message invalid account		fail	
	Blank		Error message TAN value cannot be blank			
	—	—	—			
		test maximum coverage				
6	enter the value for "**Amount**"					
	valid	5000, 5001,9000,84	**Accpect**	As expected	pass	XXX
	invalid	4999,,9001	error message amount shoulb be between (5000-9000)		fail	
7	Enter the value for "**FAN, TAN, Amount**" click on the "**Transfer**" button					
	FAN	1234				
	TAN	4321	"**Confimation Message**" amount transfer sucessfully must be displayed	As expected	pass	XXX
	Amount	6000				
8	Enter the value for "**FAN, TAN, Amount**" click on the "**Cancel**" button					
	FAN	1234				
	TAN	4321	**All field must be cleared**	As expected	pass	XXX
	Amount	6000				
Author	Sem					
Date	4/1/2020					
Reviewd by	jessica					
Approved by	ryan					

If we are saying (5000-9000) balance, but if want to test for 9001, so obiously it will give the error message (unsufficent message)

Integration test case

In this, we should not write something which we already covered in the functional test cases, and something we have written in the integration test case should not be written in the system test case again.

Rules to write integration test cases

- Firstly, understand the product
- Identify the possible scenarios
- Write the test case based on the priority

When the test engineer writing the test cases, they may need to consider the following aspects:

If the test cases are in details:

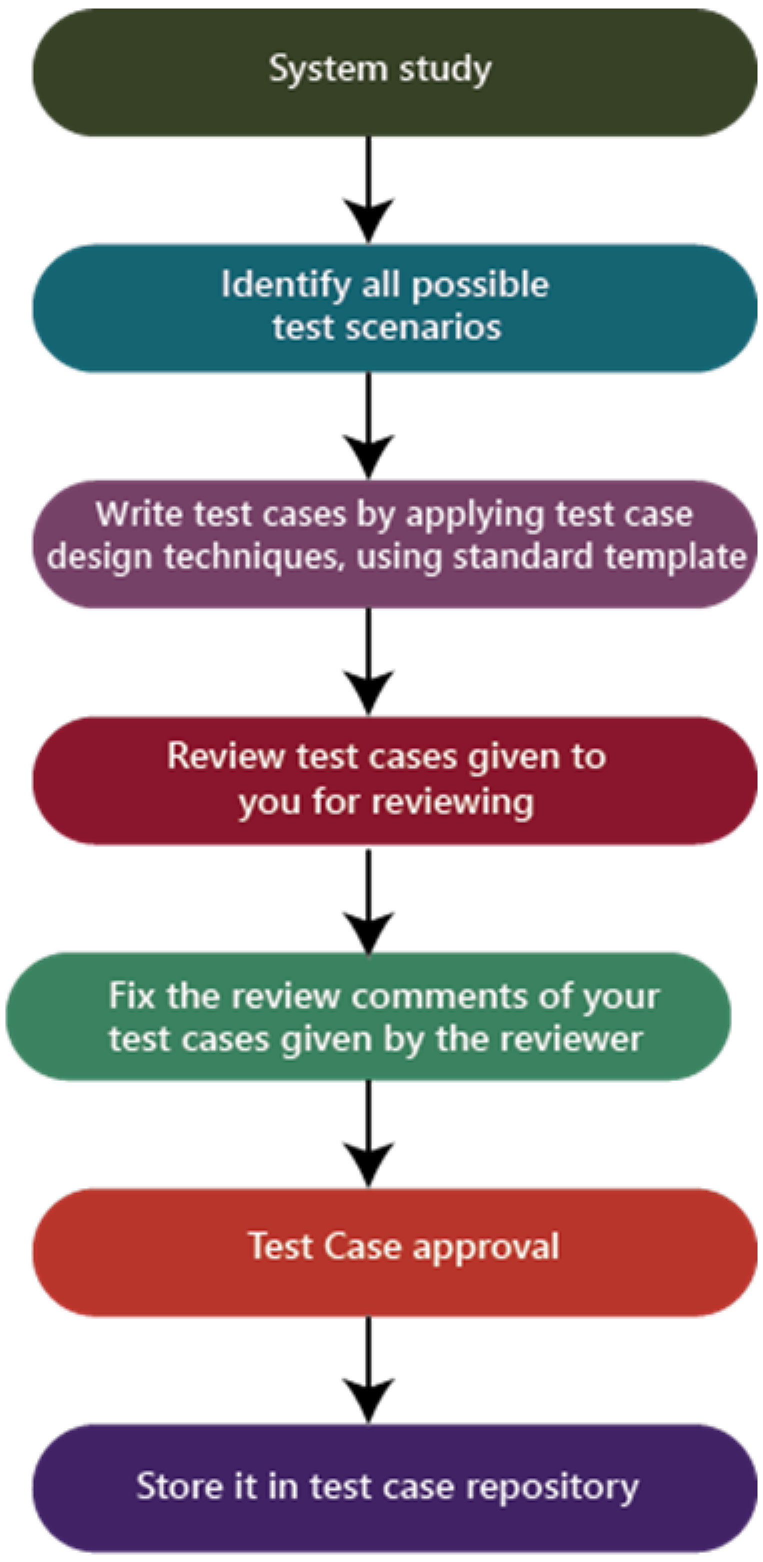
- They will try to achieve maximum test coverage.
- All test case values or scenarios are correctly described.
- They will try to think about the execution point of view.
- The template which is used to write the test case must be unique.

System test cases

We will write the system test cases for the end-to-end business flows. And we have the entire modules ready to write the system test cases.

The process to write test cases

The method of writing a test case can be completed into the following steps, which are as below:


System study
Identify all possible test scenarios
Write test cases by applying test case design techniques, using standard template
Review test cases given to you for reviewing
Fix the review comments of your test cases given by the reviewer
Test Case approval
Store it in test case repository

System study

In this, we will understand the application by looking at the requirements or the SRS, which is given by the customer.

Identify all scenarios:

- When the product is launched, what are the possible ways the end-user may use the software to identify all the possible ways.
- I have documented all possible scenarios in a document, which is called test design/high-level design.
- The test design is a record having all the possible scenarios.

Write test cases

Convert all the identified scenarios to test claims and group the scenarios related to their features, prioritize the module, and write test cases by applying test case design techniques and use the standard test case template, which means that the one which is decided for the project.

Review the test cases

Review the test case by giving it to the head of the team and, after that, fix the review feedback given by the reviewer.

Test case approval

After fixing the test case based on the feedback, send it again for the approval.

Store in the test case repository

After the approval of the particular test case, store in the familiar place that is known as the test case repository.

TEST basis

Test Basis

Test basis is defined as the source of information or the document that is needed to write test cases and also for test analysis.

Test basis should be well defined and adequately structured so that one can easily identify test conditions from which test cases can be derived.

Typical Test Basis:

- Requirement document
- Test Plan
- Codes Repository
- Business Requirement

Test suite

Test suites are the logical grouping or collection of test cases to run a single job with different test scenarios.

For instance, a test suite for product purchase has multiple test cases, like:

- Test Case 1: Login
- Test Case 2: Adding Products
- Test Case 3: Checkout
- Test Case 4: Logout

A test suite also acts as a container for test cases. It also has multiple stages for specifying the status of the test execution process, like in-progress, active, and completed. It is also known as the validations suite, with detailed information and objectives for different test cases and system configurations required for testing.

Once you create a test plan, test suites are created, which can have multiple test cases.

Test scenario

Test Scenarios are created to ensure that every functionality a website or app offers is working as expected. It is best to gather input from clients, stakeholders, and developers to create real/accurate test scenarios. This helps effectively cover all possible user scenarios and enables comprehensive testing of all business flows of the software in question.

Test Scenarios are required to verify the entire system's performance from the users' perspective. When creating them, testers need to place themselves in the users' shoes to clarify what real-world scenarios the software will have to handle when made public.

HOW TO CREATE test scenario:

1. Carefully study the Requirement Document – Business Requirement Specification (BRS), Software Requirement Specification (SRS), and Functional Requirement Specification (FRS) about the System Under Test (SUT).
2. Isolate every requirement, and identify what possible user actions need to be tested.
3. Figure out the technical issues associated with the requirement.

4. Also, remember to analyze and frame possible system abuse scenarios by evaluating the software with a hacker’s eyes.
5. Enumerate test scenarios that cover every possible feature of the software.
6. Ensure that these scenarios cover every user flow and business flow involved in the operation of the website or app.
7. After listing the test scenarios, create a Traceability Matrix to ensure every requirement is mapped to a test scenario.
8. Get the scenarios reviewed by a supervisor, and then push them to be reviewed by other stakeholders involved in the project.

Difference between Test Case and Test Scenario

Test Case	**Test Scenario**
Offers detailed information on what to test, steps required for testing, and the accurate result to be expected	Only detail information on what featur be tested and the user story associated the feature.
Required keeping testers and developers in sync	Required so that testers know what the is on a high level
It consists of low-level, individual actions testers have to undertake	Consists of high-level information (us one-liner) about what feature should b
It is derived from test scenarios	It is derived from the requirements do

Creating test cases is a one-time effort since test cases can be reused, especially during regression testing.	Test scenarios may need to be changed software evolves to align with newly developed features.
Mostly helpful for guiding individual testers on how to progress in a certain project	Most helpful in reducing complexity b listing out everything that must be test helping testers create test cases for eac scenario

METHODS OF TESTING:

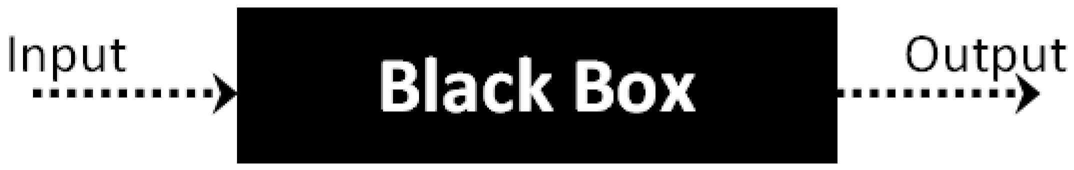
Black box testing.

White box Testing

Grey box Testing.

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.

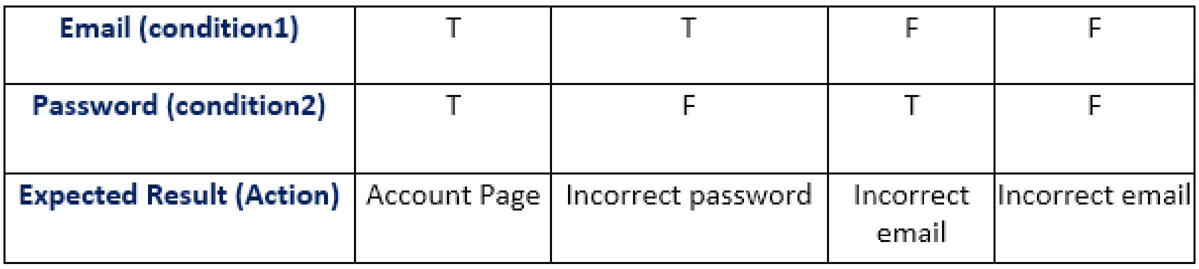


Techniques Used in Black Box Testing:

1.Decision Table Technique:

- Decision table technique is one of the widely used case **design techniques** for black box testing.
- it is also known as a cause-effect table.
- Decision table technique is appropriate for the functions that have a logical relationship between two and more than two inputs**.**
- we need to consider conditions as input and actions as output.

Let's understand it by an example:

- 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."
- Now, let's see how a decision table is created for the login function in which we can log in by using email and password. Both the email and the password are the conditions, and the expected result is action.
- 

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

- In the table, there are four conditions or test cases to test the login function. In the first condition if both email and password are correct, then the user should be directed to account's Homepage.
- In the second condition if the email is correct, but the password is incorrect then the function should display Incorrect Password. In the third condition if the email is incorrect, but the password is correct, then it should display Incorrect Email.
- Now, in fourth and last condition both email and password are incorrect then the function should display Incorrect Email.
- In this example, all possible conditions or test cases have been included, and in the same way, the testing team also includes all possible test cases so that upcoming bugs can be cured at testing level.

- ✓ In order to find the number of all possible conditions, tester uses 2^n formula where n denotes the number of inputs; in the example there is the number of inputs is 2 (one is true and second is false).
- ✓ **Number of possible conditions = 2^ Number of Values of the second condition**
  Number of possible conditions =2^2 = 4
- ✓ While using the decision table technique, a tester determines the expected output, if the function produces expected output, then it is passed in testing, and if not then it is failed. Failed software is sent back to the development team to fix the defect.

All-pairs Testing

- All-pairs testing technique is also known as pairwise testing. It is used to test all the possible discrete combinations of values.
- This combinational method is used for testing the application that uses checkbox input, radio button input (radio button is used when you have to select only one option for example when you select gender male or female, you can select only one option), list box, text box, etc.

For example:

he input values are given below that can be accepted by the fields of the given function.

1. Check Box - Checked or Unchecked
2. List Box - 0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
3. Radio Button - On or Off
4. Text Box - Number of alphabets between 1 to 100.
5. OK - Does not accept any value, only redirects to the next page.

Calculation of all the possible combinations:

1. Check Box = 2
2. List Box = 10
3. Radio Button = 2
4. Text Box = 100
5. Total number of test cases = 2*10*2*100
6. = 4000

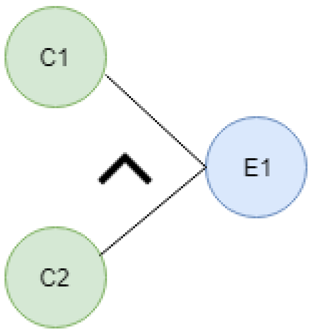
Cause and Effect Graph in Black box Testing:

Cause-effect graph comes under the black box testing technique which underlines the relationship between a given result and all the factors affecting the result. It is used to write dynamic test cases.

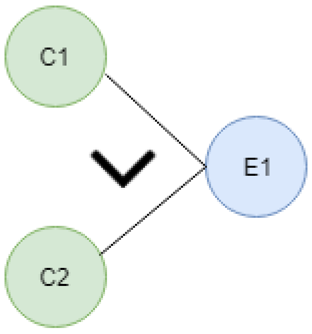
For example

For example, while using email account, on entering valid email, the system accepts it but, when you enter invalid email, it throws an error message. In this technique, the input conditions are assigned with causes and the result of these input conditions with effects.

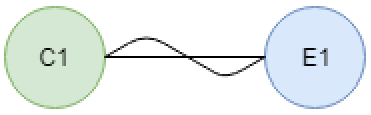
AND - E1 is an effect and C1 and C2 are the causes. If both C1 and C2 are true, then effect E1 will be true.



OR - If any cause from C1 and C2 is true, then effect E1 will be true.



NOT - If cause C1 is false, then effect E1 will be true.



Boundary Value Analysis:

It is used to test boundary values because the input values near the boundary have higher chances of error.

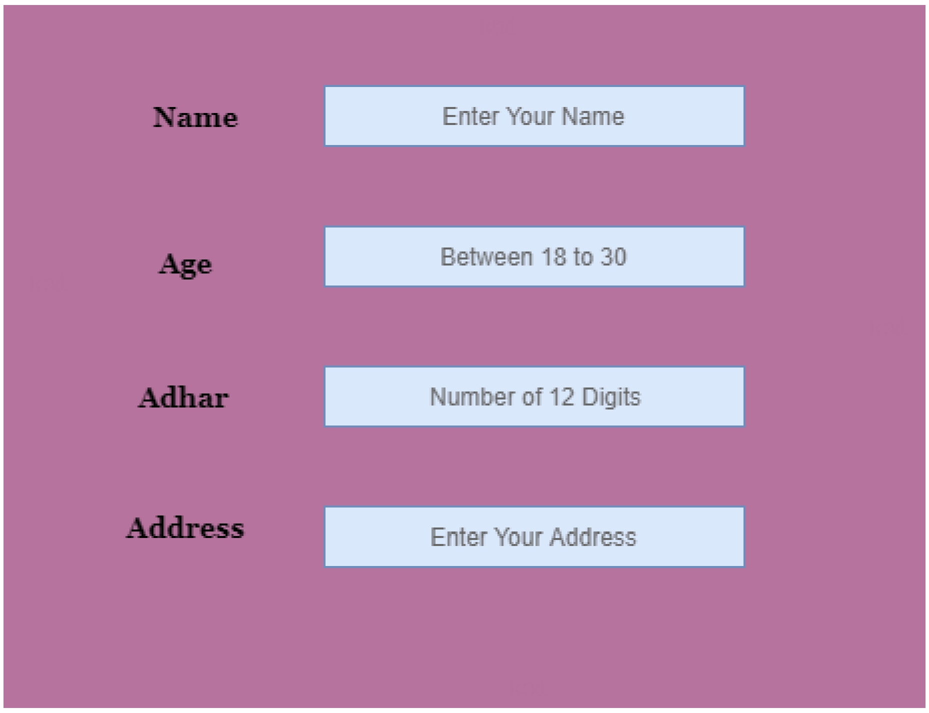
Whenever we do the testing by boundary value analysis, the tester focuses on, while entering boundary value whether the software is producing correct output or not.

For example:

Boundary values are those that contain the upper and lower limit of a variable. Assume that, age is a variable of any function, and its minimum value is 18 and the maximum value is 30, both 18 and 30 will be considered as boundary values.

The basic assumption of boundary value analysis is, the test cases that are created using boundary values are most likely to cause an error.

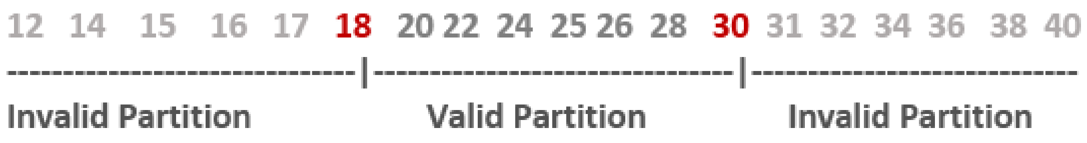
There is 18 and 30 are the boundary values that's why tester pays more attention to these values, but this doesn't mean that the middle values like 19, 20, 21, 27, 29 are ignored. Test cases are developed for each and every value of the range.

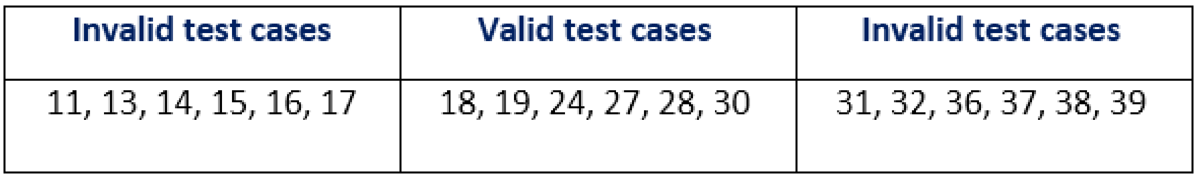


Testing of boundary values is done by making valid and invalid partitions. Invalid partitions are tested because testing of output in adverse condition is also essential.

Let's understand via practical:

Imagine, there is a function that accepts a number between 18 to 30, where 18 is the minimum and 30 is the maximum value of valid partition, the other values of this partition are 19, 20, 21, 22, 23, 24, 25, 26, 27, 28 and 29. The invalid partition consists of the numbers which are less than 18 such as 12, 14, 15, 16 and 17, and more than 30 such as 31, 32, 34, 36 and 40. Tester develops test cases for both valid and invalid partitions to capture the behavior of the system on different input conditions.





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

The software system will be passed in the test if it accepts a valid number and gives the desired output, if it is not, then it is unsuccessful. In another scenario, the software system should not accept invalid numbers, and if the entered number is invalid, then it should display error massage.

If the software which is under test, follows all the testing guidelines and specifications then it is sent to the releasing team otherwise to the development team to fix the defects.

State Transition Technique:

The general meaning of state transition is, different forms of the same situation, and according to the meaning, the state transition method does the same. It is **used to capture the behavior of the software application** when different input values are given to the same function.

For example:

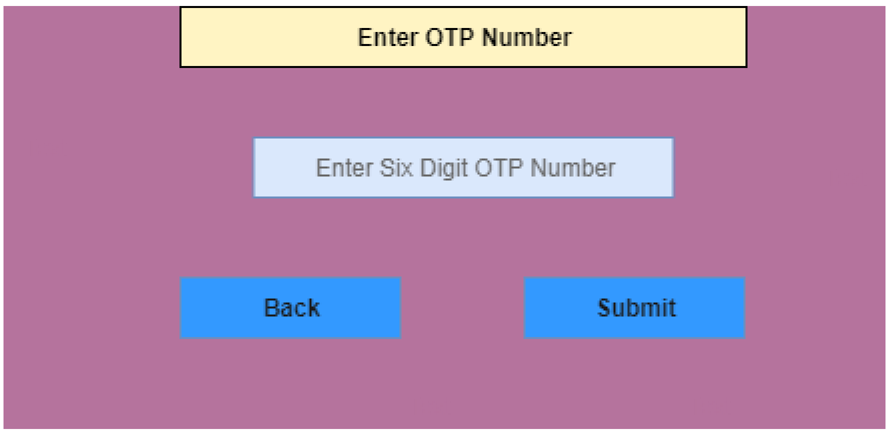
- ✓ We all use the ATMs, when we withdraw money from it, it displays account details at last.
- ✓ Now we again do another transaction, then it again displays account details, but the details displayed after the second transaction are different from the first transaction, but both details are displayed by using the same function of the ATM.
- ✓ So the same function was used here but each time the output was different, this is called state transition.

EQUIVALANCE PARTITIONING:

The principle of equivalence partitioning is, test cases should be designed to cover each partition at least once. Each value of every equal partition must exhibit the same behavior as other.

FOR EXAMPLE

1. 1. OTP Number = 6 digits



INVALID 1 Test case	**INVALID** 2 Test case	**VALID** 3 Test case	**VALID**
DIGITS >=7	DIGITS<=5	DIGITS = 6	DIGITS = 6
93847262	9845	456234	451483

Error Guessing Technique

The test case design technique or methods or approaches that need to be followed by every test engineer while writing the test cases to achieve the maximum test coverage. If we follow the test case design technique, then it became process-oriented rather than person-oriented.

Error guessing is a technique in which there is no specific method for identifying the error. It is based on the experience of the test analyst, where the tester uses the experience to guess the problematic areas of the software. It is a type of black box testing technique which does not have any defined structure to find the error.

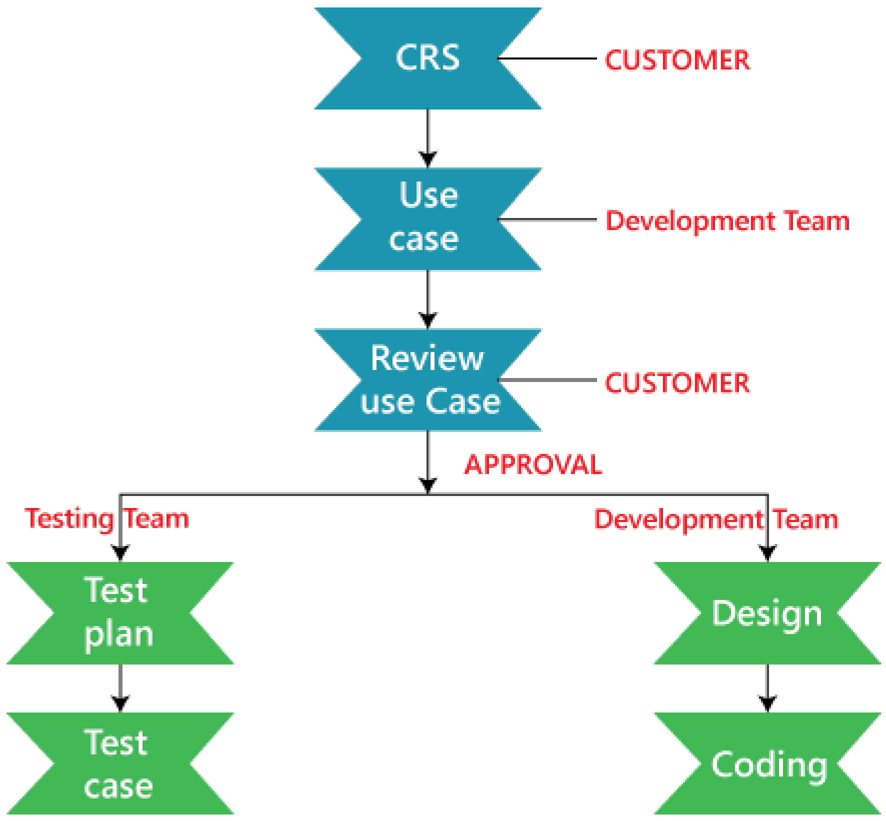
The main purpose of this technique is to identify common errors at any level of testing by exercising the following tasks:

- Enter blank space into the text fields.
- Null pointer exception.
- Enter invalid parameters.
- Divide by zero.
- Use maximum limit of files to be uploaded.
- Check buttons without entering values.

Use Case Technique

The use case is functional testing of the black box testing used to identify the test cases from the beginning to the end of the system as per the usage of the system. By using this technique, the test team creates a test scenario that can exercise the entire software based on the functionality of each function from start to end.

The client provides the customer requirement specification for the application, then the development team will write the **use case** according to the CRS, and the use case is sent to the customer for their review.



Difference between use case and prototype

Use case	Prototype
With the help of the use case, we get to know how the product should work. And it is a graphical representation of the software and its multiple features and also how they should work.	In this, we will not see how the end-user application because it is just a dummy (partic software) of the application.

A food delivery service mobile app

In this use case scenario, a food delivery mobile application wants to expand to include more food and drink establishments, even if some locations have a limited menu.

Deliver the Good Eats, a food delivery service, wants to grow the number of offered establishments and aims to include coffee shops and convenience stores. The software developers need to determine how the newly featured establishments benefit from current software parameters and what user thresholds might prompt the software through to the next stage. The team runs use cases like:

A customer searching for a specific name brand item not found in the area or chosen establishment

A customer with a low dollar amount total prompting for a minimum purchase message

A feature to allow customers to click "Order again," getting a previously purchased selection delivered again with quick user interaction

WHITE BOX TESTING

Testing technique in which software's internal structure, design, and coding are tested to verify input-output flow and improve design, usability, and security.

In white box testing, code is visible to testers, so it is also called **Clear box testing, Open box testing, Transparent box testing, Code-based testing, and Glass box testing.**

After the programming only white box can be started. Some white box tools are

- RCUNIT
- NUNIT
- CPPUNIT
- VERA CODE

ITS BASED ON STRUCTRAL BEHAVIOUR CLASSIFIED INTO WHITE BOX DESIGN TECHNIQUE:

- STATEMENT COVERAGE.
- DECISION COVERAGE.
- CONTROL FLOW.
- BRANCH COVERAGE
- PATH COVERAGE.

STATEMENT COVERAGE:

Statement coverage technique is used to design white box test cases. This technique involves execution of all statements of the source code at least once. It is used to calculate the total number of executed statements in the source code out of total statements present in the source code.

$$\text{Statement coverage} = \frac{\text{Number of executed statements}}{\text{Total number of statements}} * 100$$

EXAMPLE:

Scenario 1:
If a = 5, b = 4

```
print (int a, int b) {
int sum = a+b;
if (sum>0)
print ("This is a positive result")
else
print ("This is negative result")
}
```

Total number of statements = 7

Number of executed statements = 5

Statement coverage= 5/7*100

=500/7 =71.4%.

2.DECISION COVERAGE:

Decision coverage technique comes under white box testing which gives decision coverage to Boolean values. This technique reports true and false outcomes of Boolean expressions.

Whenever there is a possibility of two or more outcomes from the statements like **do while statement, if statement and case statement** (Control flow statements), it is considered as decision point because there are two outcomes either true or false

$$\textit{Decision Coverage} = \frac{\text{Number of Decision Outcomes Exercised}}{\text{Total Number of Decision Outcomes}} * 100$$

EXAMPLE:

Scenario 1:
If a = 5, b = 4

```
print (int a, int b) {
int sum = a+b;
if (sum>0)
print ("This is a positive result")
else
print ("This is negative result")
}
```

Total number of Decision outcomes=2

Number of Decision outcomes Exercised=1

Decision coverage =1/2*100

=0.5*100 =50%.

3.Control flow

The aim of this technique is to determine the execution order of statements or instructions of the program through a control structure.

a particular part of a large program is selected by the tester to set the testing path. It is mostly used in unit testing.

Control Flow Graph is formed from the node, edge, decision node, junction node to specify all possible execution path.

Notations used for Control Flow Graph

1. Node.

2. Edge.
3. Decision Node.
4. Junction node.

Node:

Nodes in the control flow graph are used to create a path of procedures. Basically, it represents the sequence of procedures which procedure is next to come so, the tester can determine the sequence of occurrence of procedures.

Node represent the shape: ▭ ▢

Edge:

Edge in control flow graph is used to link the direction of nodes.

We can see below in example all arrows are used to link the nodes in an appropriate direction.

Decision Node.

Decision node in the control flow graph is used to decide next node of procedure as per the value.

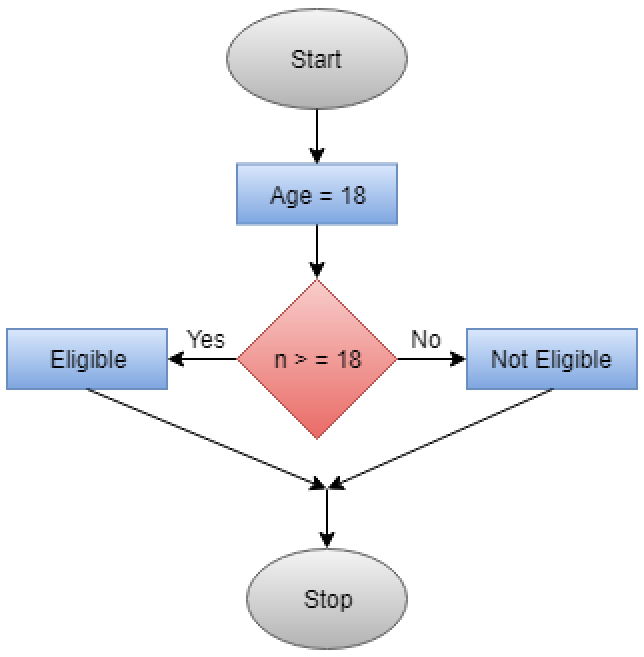
Junction node:

Junction node in control flow graph is the point where at least three links meet.

```
public class VoteEligiblityAge{



public static void main(String []args){

int n=45;

if(n>=18)

{

    System.out.println("You are eligible for voting");
```

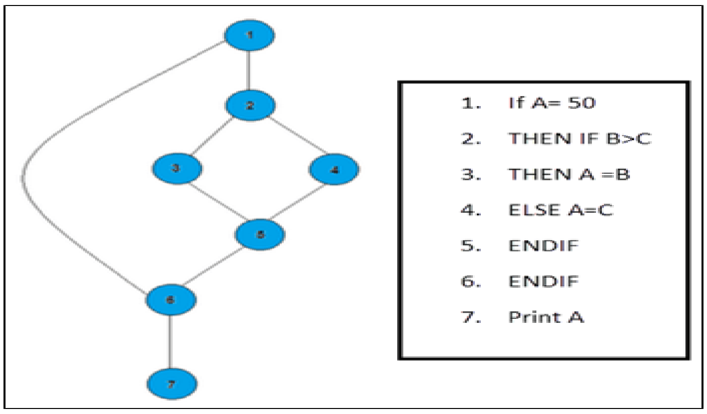
```
 }  else

{

   System.out.println("You are not eligible for voting");

 }

}

}
```

Control flow graph:



Path testing:

Path testing is a structural testing method that involves using the source code of a program in order to find every possible executable path.



```
1.   If A= 50
2.   THEN IF B>C
3.   THEN A =B
4.   ELSE A=C
5.   ENDIF
6.   ENDIF
7.   Print A
```

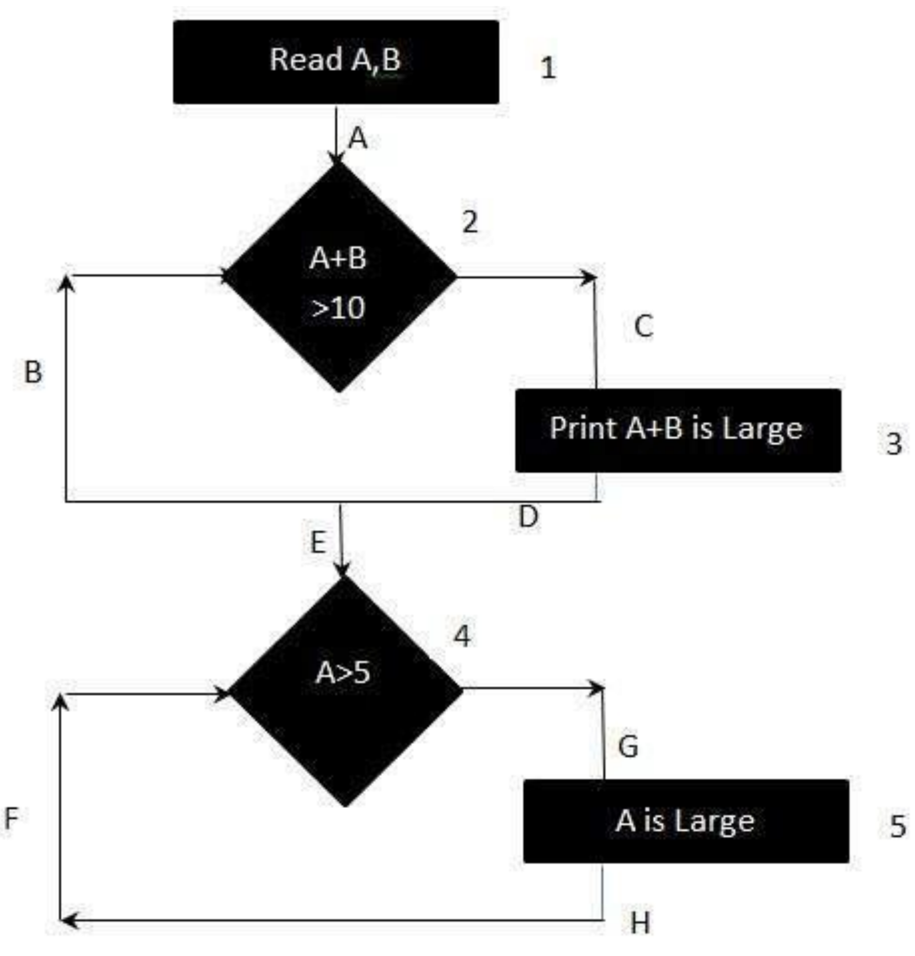
Path 1: 1,6,7.

Path 2: 1,2,3,5,6,7.

Path 3: 1,2,4,5,6,7.

Branch Coverage:

Branch coverage technique is used to cover all branches of the control flow graph. It covers all the possible outcomes (true and false) of each condition of decision point at least once.



PATH 1: A1-C2-D3-E-G4-5H.

PATH 2: A1-B2-E-F4

Hence the branch coverage is 2.

CYCLOMATIC COMPLEXITY:

It is a software metric used to indicate the complexity of a program.

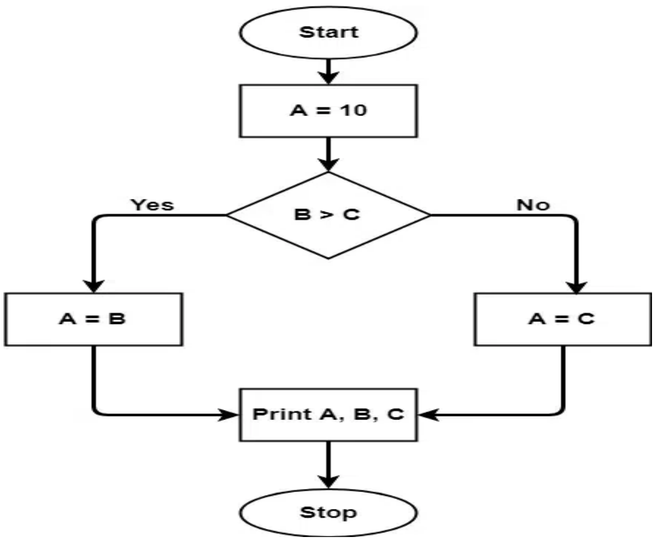
It's determine the level of confidence and stability of the software.

the quantitative measure of the number of linearly independent paths in it.

Example:

```
A = 10
  IF B > C THEN
    A = B
  ELSE
    A = C
  ENDIF
Print A
Print B
Print C
```

Control flow graph:



Cyclometric complecity= E-N+2P.

E=> EDGES =7

N=> NODES=7

P=Exit point always =1.

=7-7+2 =2

Case-2

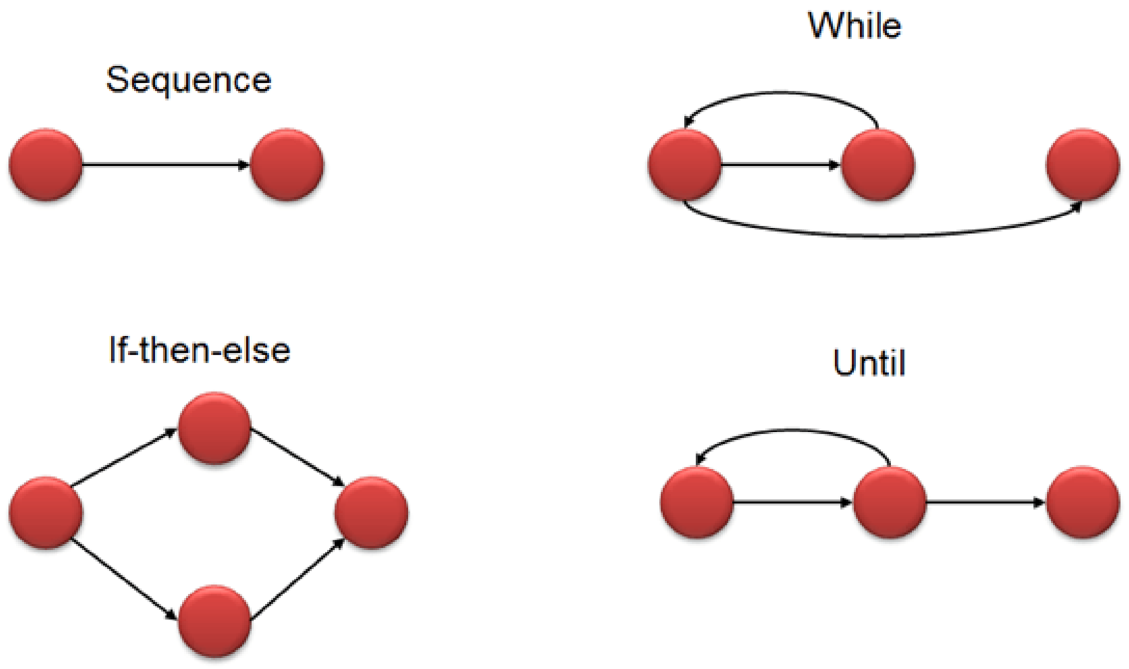
V(G)=total number of closed regions in control flow graph+1.

Case-3

V(G)=p+1

P=> predicate nodes contained.

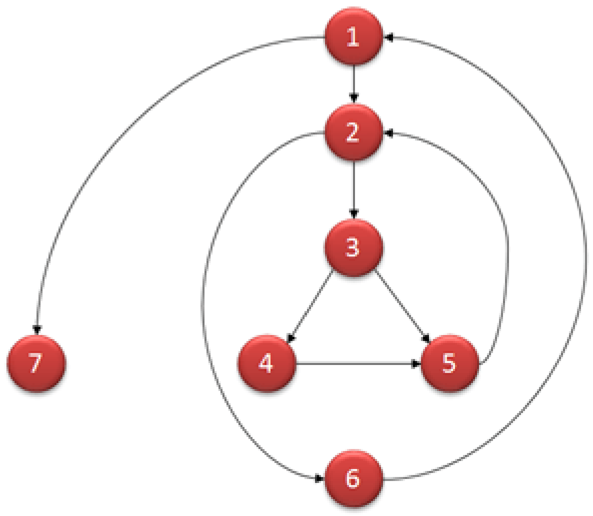
Sequence of v(g)



```
i = 0;
```

```
n=4; //N-Number of nodes present in the graph

while (i<n-1) do

j = i + 1;

while (j<n) do

if A[i]<A[j] then

swap(A[i], A[j]);

end do;

j=j+1;

end do;
```

Flow graph:



Computing mathematically,

- V(G) = 9 – 7 + 2 = 4
- V(G) = 3 + 1 = 4 (Condition nodes are 1,2 and 3 nodes)
- Basis Set – A set of possible execution path of a program
- 1, 7
- 1, 2, 6, 1, 7
- 1, 2, 3, 4, 5, 2, 6, 1, 7
- 1, 2, 3, 5, 2, 6, 1, 7