Software Testing

1. Definition
2. Verification & Validation
3. Testing Principles
4. Importance of Testing
5. Benefits of Testing
6. Levels of Testing
   - 6.1. Unit Testing
   - 6.2. Integration Testing
   - 6.3. System Testing
   - 6.4. Acceptance Testing
7. Testing Objectives
8. Software Testing Life Cycle
9. Types of Testing
   - 9.1. Manual Testing
   - 9.2. Automation Testing
10. Types of Manual Testing
    - 10.1. White Box Testing
    - 10.2. Black Box Testing
    - 10.3. Grey Box Testing
11. Types of Black Box Testing
    - 11.1. Functional Testing
    - 11.2. Non-Functional Testing
12. Black box testing Vs. White box testing
13. Black Box Techniques
14. White Box Techniques
15. Alpha and Beta Testing
16. Unit Testing
17. Integration Testing
18. System Testing
19. Debugging
20. Software Testing Strategies

Software Testing is a method to check whether the actual software product matches expected requirements and to ensure that software product is Defect free. It involves execution of software/system components using manual or automated tools to evaluate one or more properties of interest. The purpose of software testing is to identify errors, gaps or missing requirements in contrast to actual requirements.

Testing is the process of executing a program to find errors. To make our software perform well it should be error-free. If testing is done successfully it will remove all the errors from the software.

Software testing can be stated as the process of verifying and validating whether a software or application is bug-free, meets the technical requirements as guided by its design and development, and meets the user requirements effectively and efficiently by handling all the exceptional and boundary cases.

The process of software testing aims not only at finding faults in the existing software but also at finding measures to improve the software in terms of efficiency, accuracy, and usability. It mainly aims at measuring the specification, functionality, and performance of a software program or application.

Software testing can be divided into two steps:

1. Verification:

Verification is the process of evaluating a system or component at the end of a phase to determine if it satisfies the conditions imposed at the start of that phase, or, in other words, **"Are we building the product right?"**

or

Verification is the process of confirming if the software is meeting the business requirements, and is developed adhering to the proper specifications and methodologies.

- Verification ensures the product being developed is according to design specifications.
- Verification answers the question– "Are we developing this product by firmly following all design specifications?"
- Verification concentrates on the design and system specifications.

2. Validation

Validation is the process of evaluating a system or component during or at the end of the development process to determine if it satisfies the requirements of the system, or, in other words, **"Are we building the right product?"**

or

Validation is process of examining whether or not the software satisfies the user requirements. It is carried out at the end of the SDLC. If the software matches requirements for which it was made, it is validated.

- Validation ensures the product under development is as per the user requirements.
- Validation answers the question – "Are we developing the product which attempts all that user needs from this software?"
- Validation emphasizes on user requirements.

Verification	Validation
1. Verification is a static practice of verifying documents, design, code and program.	1. Validation is a dynamic mechanism of validating and testing the actual product.
2. It does not involve executing the code.	2. It always involves executing the code.
3. It is human based checking of documents and files.	3. It is computer based execution of program.
4. Verification uses methods like inspections, reviews, walkthroughs, and Desk-checking etc.	4. Validation uses methods like black box (functional) testing, gray box testing, and white box (structural) testing etc.
5. **Verification** is to check whether the software conforms to specifications.	5. **Validation** is to check whether software meets the customer expectations and requirements.
6. It can catch errors that validation cannot catch. It is low level exercise.	6. It can catch errors that verification cannot catch. It is High Level Exercise.
7. Target is requirements specification, application and software architecture, high level, complete design, and database design etc.	7. Target is actual product-a unit, a module, a bent of integrated modules, and effective final product.
8. Verification is done by QA team to ensure that the software is as per the specifications in the SRS document.	8. Validation is carried out with the involvement of testing team.
9. It generally comes first-done before validation.	9. It generally follows after **verification**.

To be more specific, software testing means that executing a program or its components in order to assure:

- The correctness of software with respect to requirements or intent;
- The performance of software under various conditions;
- The robustness of software, that is, its ability to handle erroneous inputs and unanticipated conditions;
- The usability of software under various conditions;
- The reliability, availability, survivability or other dependability measures of software; or
- Installability and other facets of a software release.

The purpose of testing is to show that the program has errors. The aim of most testing methods is to systematically and actively locate faults in the program and repair them.

Software testing is a process of identifying the correctness of software by considering its all attributes (Reliability, Scalability, Portability, Re-usability, Usability) and evaluating the execution of software components to find the software bugs or errors or defects.

Software testing provides an independent view and objective of the software and gives surety of fitness of the software. It involves testing of all components under the required services to confirm that whether it is satisfying the specified requirements or not. The process is also providing the client with information about the quality of the software.

Testing is mandatory because it will be a dangerous situation if the software fails any of time due to lack of testing. So, without testing software cannot be deployed to the end user.

Target of the test are -

- **Errors -** These are actual coding mistakes made by developers. In addition, there is a difference in output of software and desired output, is considered as an error.
- **Fault -** When error exists fault occurs. A fault, also known as a bug, is a result of an error which can cause system to fail.
- **Failure -** failure is said to be the inability of the system to perform the desired task. Failure occurs when fault exists in the system.

Why Software Testing is Important?

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

What is the need of Testing?

Testing is important because software bugs could be expensive or even dangerous. Software bugs can potentially cause monetary and human loss, and history is full of such examples.

- In April 2015, Bloomberg terminal in London crashed due to software glitch affected more than 300,000 traders on financial markets. It forced the government to postpone a 3bn pound debt sale.
- Nissan cars recalled over 1 million cars from the market due to software failure in the airbag sensory detectors. There has been reported two accident due to this software failure.
- Starbucks was forced to close about 60 percent of stores in the U.S and Canada due to software failure in its POS system. At one point, the store served coffee for free as they were unable to process the transaction.
- Some of Amazon's third-party retailers saw their product price is reduced to 1p due to a software glitch. They were left with heavy losses.
- Vulnerability in Windows 10. This bug enables users to escape from security sandboxes through a flaw in the win32k system.
- In 2015 fighter plane F-35 fell victim to a software bug, making it unable to detect targets correctly.
- China Airlines Airbus A300 crashed due to a software bug on April 26, 1994, killing 264 innocents live
- In 1985, Canada's Therac-25 radiation therapy machine malfunctioned due to software bug and delivered lethal radiation doses to patients, leaving 3 people dead and critically injuring 3 others.
- In April of 1999, a software bug caused the failure of a $1.2 billion military satellite launch, the costliest accident in history
- In May of 1996, a software bug caused the bank accounts of 823 customers of a major U.S. bank to be credited with 920 million US dollars.

Reasons why software testing techniques should be incorporated into application development:

- **Identifies defects early.**

Developing complex applications can leave room for errors. Software testing is imperative, as it identifies any issues and defects with the written code so they can be fixed before the software product is delivered.

- **Improves product quality.**

When it comes to customer appeal, delivering a quality product is an important metric to consider. An exceptional product can only be delivered if it's tested effectively before launch. Software testing helps the product pass quality assurance (QA) and meet the criteria and specifications defined by the users.

- **Increases customer trust and satisfaction.**

Testing a product throughout its development lifecycle builds customer trust and satisfaction, as it provides visibility into the product's strong and weak points. By the time customers receive the product, it has been tried and tested multiple times and delivers on quality.

- **Detects security vulnerabilities.**

Insecure application code can leave vulnerabilities that attackers can exploit. Since most applications are online today, they can be a leading vector for cyber attacks and should be tested thoroughly during various stages of application development. For example, a web application published without proper software testing can easily fall victim to a cross-site scripting attack where the attackers try to inject malicious code into the user's web browser by gaining access through the vulnerable web application. The nontested application thus becomes the vehicle for delivering the malicious code, which could have been prevented with proper software testing.

- **Helps with scalability**.

A type of nonfunctional software testing process, scalability testing is done to gauge how well an application scales with increasing workloads, such as user traffic, data volume and transaction counts. It can also identify the point where an application might stop functioning and the reasons behind it, which may include meeting or exceeding a certain threshold, such as the total number of concurrent app users.

- **Saves money.**

Software development issues that go unnoticed due to a lack of software testing can haunt organizations later with a bigger price tag. After the application launches, it can be more difficult to trace and resolve the issues, as software patching is generally more expensive than testing during the development stages.

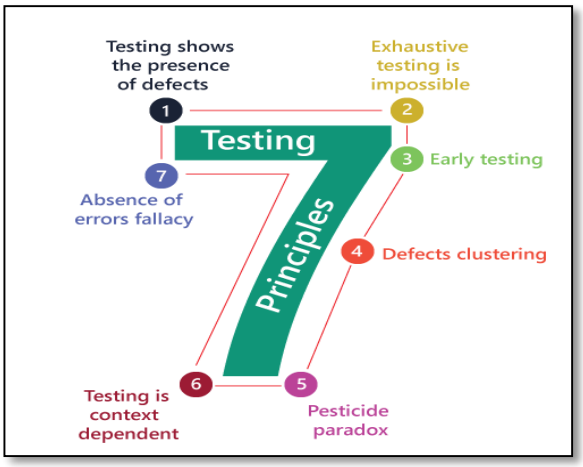
What are the benefits of Software Testing?

Here are the benefits of using software testing:

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

Principles of Testing:-

Software testing is a procedure of implementing software or the application to identify the defects or bugs. For testing an application or software, we need to follow some principles to make our product defects free, and that also helps the test engineers to test the software with their effort and time.



Seven different testing principles are:

- Testing shows the presence of defects
- Exhaustive Testing is not possible
- Early Testing
- Defect Clustering
- Pesticide Paradox
- Testing is context-dependent
- Absence of errors fallacy

Testing shows the presence of defects

As stated in this testing principle, "Testing talks about the presence of defects and doesn't talk about the absence of defects". In software testing, we look for bugs to be fixed before we deploy systems to live environments – this gives us confidence that our systems will work correctly when goes live to users. Despite this, the testing process does not guarantee that software is 100% error-free. It is true that testing greatly reduces the number of defects buried in software, however discovering and repairing these problems does not guarantee a bug-free product or system.

Even if testers cannot find defects after repeating regression testing, it does not mean the software is 100 % bug-free. For instance, an application may appear to be error-free after passing various stages of testing, but when it is deployed in the environment, an unexpected defect can be found. Team members should always adhere to this concept, and effort should be made to manage client expectations.

Exhaustive Testing is not possible

Exhaustive testing usually tests and verifies all functionality of a software application while using both valid and invalid inputs and pre-conditions. No matter how hard you try, testing EVERYTHING is pretty much impossible. The inputs and outputs alone have an infinite number of combinations, so it is 100% not possible to test an application from every angle.

Consider the case when we have to test an input field that accepts percentages between 50 and 55, so we test the field using 50, 51, 52, 53, 54, 55. Assuming that the same input field accepts values from 50 to 100, we will need to test using 50, 51, 52, 53, …., 99, 100. This is a basic example. You may think that an automation tool would be able to accomplish this. But imagine a field that accepts a billion values. Will it be possible to test all possible values?

As long as we continue to test all possible scenarios, the software execution time and cost will increase. In order to avoid doing exhaustive testing, we will take into consideration some important testing criteria effects such as risks and priorities as part of our testing efforts and estimates.

Hence, instead of performing the exhaustive testing, so we can complete this according to the importance of the modules because the product timelines will not permit us to perform such type of testing scenarios.

Early Testing

In software development, early testing means incorporating testing as early as possible in the development process. It plays a critical role in the software development lifecycle (SDLC). For instance, testing the requirements before coding begins. Amending issues during this stage of a project's life cycle is much cheaper and easier than amending issues at the end of the project when we must write new sections of functionality, resulting in overruns and late deadlines. The cost to fix a bug increases exponentially with time as the development life cycle progresses.

Let's consider two scenarios. In the first case, you found an incorrect requirement in the requirement gathering phase. In the second case, you found a defect in a fully developed functionality. It is less expensive to fix the incorrect requirement than fully developed functionality that isn't working the way it should. Therefore, to improve software performance, software testing should begin at the initial phase, that is, during requirement analysis.

Defect clustering

In software testing, defect clustering refers to a small module or feature that has the most bugs or operation issues. This is because defects are not evenly distributed within a system but are clustered. It could be due to multiple factors, such as the modules might be complicated or the coding related to such modules might be complex.

Pareto Principle (80-20 Rule) states that 80% of issues originate from 20% of modules, while the remaining 20% originate from the remaining 80% of modules. Thus, we prioritize testing on 20% of modules where we experience 80% of bugs.

For an effective testing strategy, it is necessary to thoroughly examine these areas of the software. The defect clustering method relies on the teams' knowledge and experience to identify which modules to test. You can identify such risky modules from your experience. Therefore, the team only has to focus on those "sensitive" areas, saving both time and effort.

Pesticide paradox

In software testing, the Pesticide Paradox generally refers to the practice of repeating the exact same test cases over and over again. As time passes, these test cases will cease to find new bugs. Developers will create tests which are passing so they can forget about negative or edge cases. This is based on the theory that when you repeatedly spray the same pesticide on crops in order to eradicate insects, the insects eventually develop an immunity, making the pesticide ineffective. The same is true for software testing.
Therefore, in order to overcome the Pesticide Paradox, it is imperative to regularly review and update the test cases so that more defects can be found. However, if this process is not followed, and the same tests are repeated over and over again, then eventually there will be no new bugs found, but it doesn't mean the system is 100 % bug free. To make testing more effective, testers must constantly look for ways to improve the existing test methods. To test new features of the software or system, new tests must be developed.

Testing is context-dependent

Each type of software system is tested differently. According to this principle, testing depends on the context of the software developed, and this is entirely true. The reality is that every application has its own unique set of requirements, so we can't put testing in a box. Of course, every application goes through a defined testing process, however, the testing approach may vary based on the application type.
Various methodologies, techniques, and types of testing are used depending on the nature of an application. For example, health industry applications require more testing than gaming applications, safety-critical systems (such as an automotive or airplane ECU) require more testing than company presentation websites, and online banking applications will require different testing approaches than e-commerce sites or advertising sites.

Absence of errors fallacy

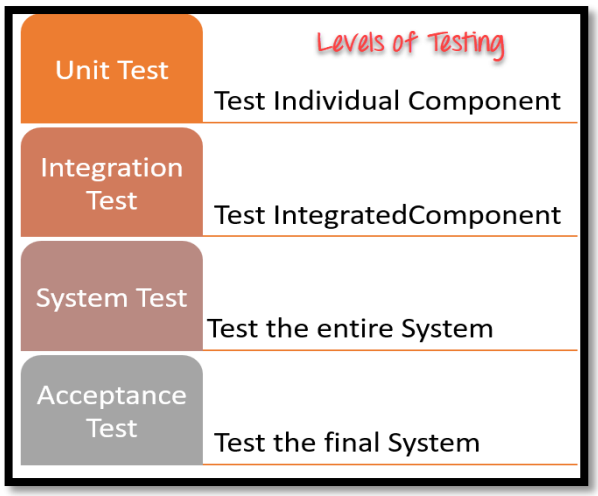
The software which we built not only must be 99% bug-free software but also it must fulfill the business, as well as user requirements otherwise it will become unusable software. Even bug-free software may still be unusable if incorrect requirements are incorporated into the software, or if the software fails to meet the business needs.
If you build it, they will come!!! There is a myth that if you build a bug-free system, users will come and use it, but this is not true. In order for software systems to be usable, it must not only be 99% bug-free software but also fulfill the business needs and user requirements. So, irrespective of how flawless or error-free a system may be, if it lacks usability and is hard to use, or if it does not match business/user needs, it is only a failure.

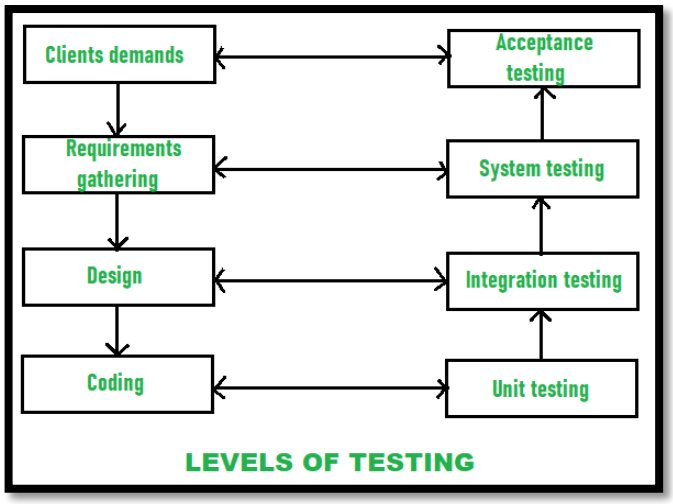
Levels of Testing

Software Testing is an activity performed to identify errors so that errors can be removed to obtain a product with greater quality. To assure and maintain the quality of software and to represents the ultimate review of specification, design, and coding, Software testing is required.
There are mainly four Levels of Testing in software testing:

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

These testing can be conducted at various stages of software development. The levels of testing along with the corresponding software development phase is shown by the following diagram –





Level1: Unit Testing (checks if software components are fulfilling functionalities or not)

- Unit testing is the first level of software testing, which is used to test if software modules are satisfying the given requirement or not. This kind of testing is performed by developers.
- The first level of testing involves analyzing each unit or an individual component of the software application. It is the smallest testable part of the software.
- Unit testing will help the test engineer and developers in order to understand the base of code that makes them able to change defect causing code quickly.
- Unit testing is also the first level of functional testing. The primary purpose of executing unit testing is to validate unit components with their performance.

Level2: Integration Testing (checks the data flow from one module to other modules)

- The second level of software testing is the integration testing. The integration testing process comes after unit testing.
- In this testing, two or more modules which are unit tested are integrated to test i.e. technique interacting components and are then verified if these integrated modules work as per the expectation or not and interface errors are also detected.
- Integrating testing checks the data flow from one module to other modules. This kind of testing is performed by testers.

Level3: System Testing (evaluates both functional and non-functional needs for the testing)

- In the third level of software testing, we will test the application as a whole system.
- System testing is performed on a complete, integrated system i.e. all the system elements forming the system is tested as a whole to meet the requirements of the system. It allows checking system's compliance as per the requirements. It tests the overall interaction of components. It involves load, performance, reliability and security testing.
- System testing most often the final test to verify that the system meets the specification. It evaluates both functional and non-functional need for the testing.
- This kind of testing is performed by a testing team.

Level4: Acceptance Testing

- The last and fourth level of software testing is acceptance testing, which is used to evaluate whether a specification or the requirements are met as per its delivery.
- It is a kind of testing conducted to ensure whether the requirement of the users are fulfilled prior to its delivery and the software works correctly in the user's working environment.
- Acceptance testing is basically done by the user or customer. However, other stockholders can be involved in this process.

Testing Objectives

Software testing is a crucial element in the software development life cycle (SDLC), which can help software engineers save time & money of organizations by finding errors and defects during the early stages of software development. With the assistance of this process one can examine various components associated with the application and guarantee their appropriateness.

The goals and objectives of software testing are numerous, which when achieved help developers build a defectless and error free software and application that has exceptional performance, quality,effectiveness, security, among other things.

Though the objective of testing can vary from company to company and project to project, there are some goals that are similar for all. These objectives are:

1. **Verification:** A prominent objective of testing is verification, which allows testers to confirm that the software meets the various business and technical requirements stated by the client before the inception of the whole project.
2. **Validation:** Confirms that the software performs as expected and as per the requirements of the clients. Validation involves checking the comparing the final output with the expected output and then making necessary changes if there is a difference between the two.
3. **Defects:** The most important purpose of testing is to find different defects in the software to prevent its failure or crash during implementation or go live of the project. Defects if left undetected or unattended can harm the functioning of the software and can lead to loss of resources, money, and reputation of the client. Therefore, software testing is executed regularly during each stage of software development to find defects of various kinds.
4. **Providing Information:** With the assistance of reports generated during the process of software testing, testers can accumulate a variety of information related to the software and the steps taken to prevent its failure. These, then can be shared with all the stakeholders of the project for better understanding of the project as well as to establish transparency between members.
5. **Preventing Defects:** During the process of testing the aim of testes to identify defects and prevent them from occurring again in the future. To accomplish this goal, software is tested rigorously by a independent testers, who are not responsible for software development.
6. **Quality Analysis:** Testing helps improve the quality of the software by constantly measuring and verifying its design and coding. Additionally, various types of testing techniques are used by testers, which help them achieve the desired software quality.
7. **Compatibility:** It helps validate application's compatibility with the implementation environment, various devices, Operating Systems, user requirements, among other things.
8. **For Optimum User Experience:** Easy software and application accessibility and optimum user experience are two important requirements that need to be accomplished for the success of any project as well as to increase the revenue of the client. Therefore, to ensure this software is tested again and again by the testers with the assistance of stress testing, load testing, spike testing, etc.
9. **Verifying Performance & Functionality:** It ensures that the software has superior performance and functionality. This is mainly verified by placing the software under extreme stress to identify and measure its all plausible failure modes. To ensure this, performance testing, usability testing, functionality testing, etc. is executed by the testers.

Software Testing Life Cycle (STLC)

The procedure of software testing is also known as STLC (Software Testing Life Cycle) which includes phases of the testing process.The testing process is executed in a well-planned and systematic manner. All activities are done to improve the quality of the software product.

Software testing life cycle contains the following steps:

1. Requirement Analysis
2. Test Plan Creation
3. Environment setup
4. Test case Execution
5. Defect Logging
6. Test Cycle Closure

1. Requirement Analysis:

The first step of the manual testing procedure is requirement analysis. In this phase, tester analyses requirement document of SDLC (Software Development Life Cycle) to examine requirements stated by the client. After examining the requirements, the tester makes a test plan to check whether the software is meeting the requirements or not.

2. Test Plan Creation:

A **Test Plan** is a detailed document that describes the test strategy, objectives, schedule, estimation, deliverables, and resources required to perform testing for a software product. Test plan creation is the crucial phase of STLC where all the testing strategies are defined. Tester determines the estimated effort and cost of the entire project. This phase takes place after the successful completion of the Requirement Analysis Phase. Testing strategy and effort estimation documents provided by this phase. Test case execution can be started after the successful completion of Test Plan Creation.

3. Environment setup:

Setup of the test environment is an independent activity and can be started along with **Test Case Development**. This is an essential part of the manual testing procedure as without environment testing is not possible. Environment setup requires a group of essential software and hardware to create a test environment. The testing team is not involved in setting up the testing environment, its senior developers who create it.

4. Test case Execution:

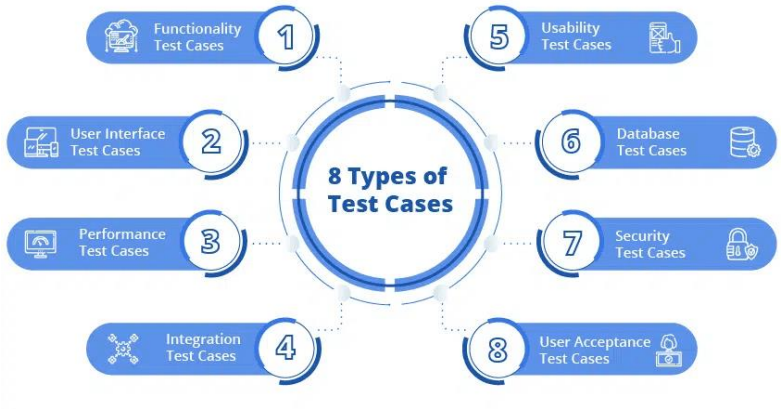
Test Case Example :

The following is an example of a test case to check the login functionality,

Test Case ID	Test Case Description	Test Steps	Test Data	Expected Result	Actual Result
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
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

Test case Execution takes place after the successful completion of test planning. In this phase, the testing team starts case development and execution activity. The testing team writes down the detailed test cases, also prepares the test data if required. The prepared test cases are reviewed by peer members of the team or Quality Assurance leader.

RTM (Requirement Traceability Matrix) is also prepared in this phase. Requirement Traceability Matrix is industry level format, used for tracking requirements. Each test case is mapped with the requirement specification. Backward & forward traceability can be done via RTM.



5. Defect Logging:

Testers and developers evaluate the completion criteria of the software based on test coverage, quality, time consumption, cost, and critical business objectives. **This phase determines the characteristics and drawbacks of the software.** Test cases and bug reports are analyzed in depth to detect the type of defect and its severity.

Defect logging analysis mainly works to find out defect distribution depending upon severity and types. If any defect is detected, then the software is returned to the development team to fix the defect, then the software is re-tested on all aspects of the testing.

Once the test cycle is fully completed then test closure report, and test metrics are prepared.

6. Test Cycle Closure:

The test cycle closure report includes all the documentation related to software design, development, testing results, and defect reports.

This phase evaluates the strategy of development, testing procedure, possible defects in order to use these practices in the future if there is a software with the same specification.

Test Plan

A test plan is a document that consists of all future testing-related activities. It is prepared at the project level and in general, it defines work products to be tested, how they will be tested, and test type distribution among the testers. Before starting testing there will be a test manager who will be preparing a test plan. In any company whenever a new project is taken up before the tester involves in the testing the test manager of the team would prepare a test Plan.

A test plan is a detailed document which describes software testing areas and activities. It outlines the test strategy, objectives, test schedule, required resources (human resources, software, and hardware), test estimation and test deliverables.

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.
The test plan is a base of every software's testing. It is the most crucial activity which ensures availability of all the lists of planned activities in an appropriate sequence.

The test plan is a template for conducting software testing activities as a defined process that is fully monitored and controlled by the testing manager. The test plan is prepared by the Test Lead (60%), Test Manager(20%), and by the test engineer(20%).

Objectives of Software Test Plan

- It provides an overview of where to start and stop the work.
- To be precise about the number of resources needed to finish the work.
- Timeline, based on the number of hours and workers needed.
- It has every detail from beginning to end, just like a prototype.
- To create the detailed tasks that must be carried out in the project's modules.
- It serves as a guide for following rules when the project is completed phase by phase.
- It will assist in considering the project's challenges and identifying solutions.
- All stakeholder interactions will be taken into consideration and organized.

Importance of Test Plan

- The test plan gives direction to our thinking (or guides 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.
- Writing a test strategy directs our thought process. Writing a test plan forces us to face the difficulties ahead and concentrate our thoughts on crucial issues.
- A communication channel is formed with other project team members, peers, testers, managers, and other stakeholders through which the test strategy and the project can impact one another. This is unquestionably true concerning organizational-wide testing policies and motives, project and product risks, test scope, resource concerns, objectives, essential areas to test, constraint considerations, and the testability of the item.
- We handle change with the use of the test plan. Early on in the project, we changed our plans as we learned new information. We modify our plans as the project progresses and the situation changes. We can keep testing in line with project requirements by revising the strategy at significant milestones. We finalize our plans as we conduct the tests and base them on the outcomes.

Types of Test Plans in Software Testing

1. Master Test Plan

(Master Test Plan is a type of test plan that has multiple levels of testing. It includes a complete test strategy.)

The master test plan is a document that goes into great depth on the planning and management of testing at various test levels. It provides a bird's eye view of the important choices made, the tactics used, and the testing effort put forth for the project. The master test plan includes the list of tests that must be run. Test coverage, connections between various test levels and associated code tasks, test execution strategies, etc.

2. Phase Test Plan

(A phase test plan is a type of test plan that addresses any one phase of the testing strategy. For example, a list of tools, a list of test cases, etc.)

The testing plans that must be followed for each test level, or occasionally test type, are described in detail in the level test plan. The level test plan typically includes further information on the levels listed in the master testing plan. They would offer the testing schedule, benchmarks, activities, templates, and other information that isn't included in the master plan.

3. Specific Test Plans

(In this type of test plan, it is designed for specific types of testing especially non-functional testing.)

Plans for conducting particular testing, such as performance and security tests. For instance, performance testing is software testing that aims to ascertain how a system responds and performs under a specific load. Security testing is software testing that aims to ascertain the system's vulnerabilities and whether its data and resources are safe from potential intruders.

Test Plan Guidelines

- Avoid Overlapping and repetition.
- Avoid Lengthy Paragraph.
- Use lists and tables.
- Update plan.
- Don't use outdated documents.

How to write a Test Plan

Follow the seven steps below to create a test plan as per IEEE 829

1. Analyze 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

Step 1) Analyze the product

There is a requirement of complete knowledge and information about the product to make the test plan.

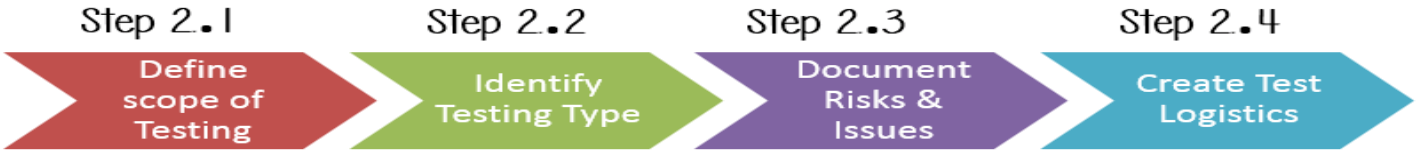
The answer is Impossible. Product must be learn throughly before testing it.

Step 2) Develop Test Strategy

Test Strategy is a **critical step** in making a Test Plan in Software Testing. A Test Strategy document is a high-level document, which is usually developed by Test Manager. This document defines:

- The project's **testing objectives** and the means to achieve them
- Determines testing **effort** and **costs**

To develop Test Strategy, follow the steps given below



Step 2.1) Define Scope of Testing

Before the start of any test activity, scope of the testing should be known. You must think hard about it.

- The components of the system to be tested (hardware, software, middleware, etc.) are defined as "**in scope**"
- The components of the system that will not be tested also need to be clearly defined as being "**out of scope**."

Defining the scope of your testing project is very important for all stakeholders. With a precise scope, all project members will have a **clear** understanding about what is tested and what is not

- ***How do you determine scope your project?***

To determine scope, you must –

- *Precise customer requirement*
- *Project Budget*
- *Product Specification*
- *Skills & talent of your test team*

Now should clearly define the "in scope" and "out of scope" of the testing.

- *As the software requirement specifications, the project ABC Bank only focus on testing all the **functions** and external interface of website **ABC** Bank (**in scope** testing)*
- *Nonfunctional testing such as **stress**, **performance** or **logical database** currently will not be tested. (**out of** scope)*

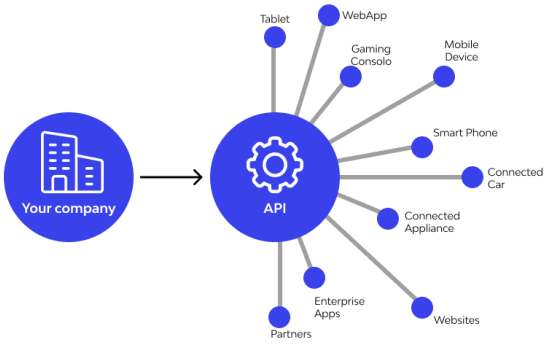
Step 2.2) Identify Testing Type

A **Testing Type** is a standard test procedure that gives an expected test outcome.

Each testing type is formulated to identify a specific type of product bugs. But, all Testing Types are aimed at achieving one common goal "**Early detection of** all the defects before releasing the product to the customer"

The **commonly used** testing types are:

- Unit Testing
- API Testing
- Integration Testing
- System Testing
- Agile Testing



There are **tons of Testing Types** for testing software product. Your team **cannot have** enough efforts to handle all kind of testing. As Test Manager, you must set **priority** of the Testing Types

Step 2.3) Document Risk & Issues

Risk is future's **uncertain event** with a probability of **occurrence** and a **potential** for loss. When the risk actually happens, it becomes the '**issue'.** In the QA Test Plan, you will document those risks.

Step 2.4) Create Test Logistics

In Test Logistics, the Test Manager should answer the following questions:

- **Who** will test?
- **When** will the test occur?

Who will test?

You may not know exact names of the tester who will test, but the **type of tester** can be defined.

To select the right member for specified task, you have to consider if his skill is qualified for the task or not, also estimate the project budget. Selecting wrong member for the task may cause the project to **fail** or **delay**.

Person having the following skills is most ideal for performing software testing:

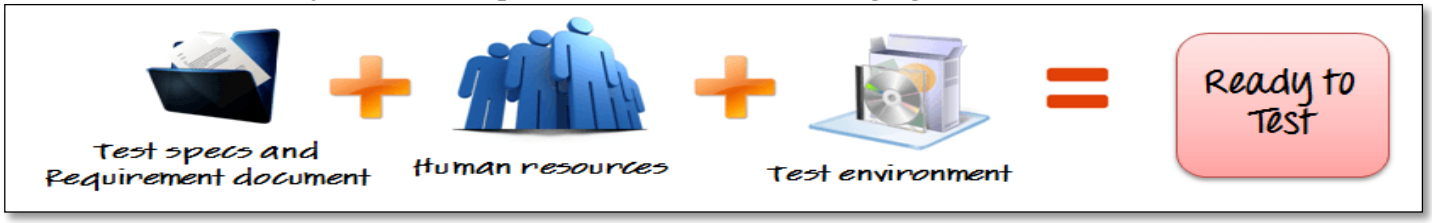
- Ability to **understand** customers point of view
- Strong **desire** for quality
- **Attention** to detail
- Good **cooperation**

In your project, the member who will take in charge for the test execution is the **tester.** Base on the project budget, you can choose in-source or outsource member as the tester.

When will the test occur?

Test activities must be matched with associated development activities.

You will start to test when you have **all required items** shown in following figure



Step 3) Define Test Objective

Test Objective is the overall goal and achievement of the test execution. The objective of the testing is finding as many software defects as possible; ensure that the software under test is **bug free** before release.
To define the test objectives, you should do 2 following steps

1. List all the software features (functionality, performance, GUI…) which may need to test.
2. Define the **target** or the **goal** of the test based on above features

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
Suspension Criteria
Specify the critical suspension criteria for a test. If the suspension criteria are met during testing, the active test cycle will be **suspended** until the criteria are **resolved**.
Test Plan 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.
Exit Criteria
It specifies the criteria that denote a **successful** completion of a test phase. The exit criteria are the targeted results of the test and are necessary before proceeding to the next phase of development. Example: **95%** of all critical test cases must pass.
Step 5) Resource Planning
Resource plan is a **detailed summary** of all types of resources required to complete project task. Resource could be human, equipment and materials needed to complete a project
The resource planning is important factor of the test planning because helps in **determining** the **number** of resources (employee, equipment…) to be used for the project. Therefore, the Test Manager can make the correct schedule & estimation for the project.
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 environment consists of real business and user environment, as well as physical environments, such as server, front end running environment.
To finish this task, you need **a strong cooperation** between Test Team and Development Team
*For Example :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

In the Test Estimation phase, suppose you break out the whole project into small tasks and add the estimation for each task. Then you create the **schedule** to complete these tasks.
Step 8) Test Deliverables
Test Deliverables is a list of all the documents, tools and other components that has to be developed and maintained in support of the testing effort.
There are different test deliverables at every phase of the software development lifecycle.

Test deliverables are provided **before** testing phase.

- Test plans document.
- Test cases documents
- Test Design specifications.

Test deliverables are provided **during** the testing

- Test Scripts
- Simulators.
- Test Data
- Test Traceability Matrix
- Error logs and execution logs.

Test deliverables are provided **after** the testing cycles is over.

- **Test Results/reports**
- Defect Report
- Installation/ Test procedures guidelines
- **Release notes**

What If There is No Test Plan?
Below are some of the situations that may occur if there is no test plan in place:

- Misunderstanding roles and responsibilities.
- The test team will have no clear objective.
- No surety when the process ends.
- Undefined test scope may confuse testers.

Test Case Specification

- Test Case Specification document described detailed summary of what scenarios will be tested, how they will be tested, how often they will be tested, and so on and so forth, for a given feature.
- It specifies the purpose of a specific test, identifies the required inputs and expected results, provides step-by-step procedures for executing the test, and outlines the pass/fail criteria for determining acceptance.
- Test Case Specification has to be done separately for each unit. However, a Test Plan is a collection of all Test Specifications for a given area. The Test Plan contains a high-level overview of what is tested for the given feature area.
- This document specifies the main objective of a specific test and identifies the required inputs as well as expected results/outputs. Moreover, it acts as a guide for executing the procedure of testing and outline the pass & fail criteria for determining acceptance.
- Test case specification is among those documents, whose format is set by IEEE Standard for Software & System Test Document (829-1998). With the assistance of test case specification document one can verify the quality of the numerous test cases created during the software testing phase.

Format for Test Case Specifications:

Developed by the organization that is responsible for formal testing of the software, the test case specification document needs to be prepared separately for each unit to ensure its effectiveness and to help build a proper and efficient test plan.
Therefore, the format that is used for creating this document is:

- ***Test Case Objectives****: Purpose of the test*
- ***Test Items****: Items (e.g., requirement specifications, design specifications, code, etc.) required to run a particular test case. This should be provided in "**Notes**" or "**Attachment**" feature. It describes the features and conditions required for testing.*
- ***Input Specifications****: Description of what is required (step-by-step) to execute the test case (e.g., input files, values that must be entered into a field, etc.). This should be provided in "**Action**" field.*
- ***Output Specifications****: Description of what the system should look like after the test case is run. This should be provided in the "**Expected Results**" field.*
- ***Environmental Needs****: Description of any special environmental needs. This includes system architectures, Hardware & Software tools, records or files, interfaces, etc.*

To sum up, ***Test Case Specification*** defines the exact set up and inputs for one Test Case.

Reason for Test Case Specification:
There are two basic reasons test cases are specified before they are used for testing:

1. Testing has severe limitations and the effectiveness of testing depends heavily on the exact nature of the test case. Even for a given criterion the exact nature of the test cases affects the effectiveness of testing.
2. Constructing a good Test Case that will reveal errors in programs is a very creative activity and depends on the tester. It is important to ensure that the set of test cases used is of high quality. This is the primary reason for having the test case specification in the form of a document.

The Test Case Specification is developed in the Development Phase by the organization responsible for the formal testing of the application.

Difference between Test Strategy and Test Plan

- Test Plan is a document that describes the scope, objective and weight on software testing task whereas Test Strategy describes how testing needs to be done.
- Test Plan is used at the project level whereas Test Strategy is used at the organization level.
- Test Plan has the primary goal of how to test, when to test and who will verify whereas Test Strategy has the primary goal of what technique to follow and which module to check.
- Test Plan can be changed whereas Test Strategy can't change.
- Test Plan is carried out by the test manager whereas the Test Strategy is carried out by the project manager.

Test Plan	Test Strategy
A test plan for software project can be defined as a document that defines the scope, objective, approach and emphasis on a software testing effort	Test strategy is a set of guidelines that explains test design and determines how testing needs to be done
Components of Test plan include- Test plan id, features to be tested, test techniques, testing tasks, features pass or fail criteria, test deliverables, responsibilities, and schedule, etc.	Components of Test strategy includes- objectives and scope, documentation formats, test processes, team reporting structure, client communication strategy, etc.
Test plan is carried out by a testing manager or lead that describes how to test, when to test, who will test and what to test	A test strategy is carried out by the project manager. It says what type of technique to follow and which module to test
Test plan narrates about the specification	Test strategy narrates about the general approaches
Test plan can change	Test strategy cannot be changed
Test planning is done to determine possible issues and dependencies in order to identify the risks.	It is a long-term plan of action. You can abstract information that is not project specific and put it into test approach
A test plan exists individually	In smaller project, test strategy is often found as a section of a test plan
It is defined at project level	It is set at organization level and can be used by multiple projects

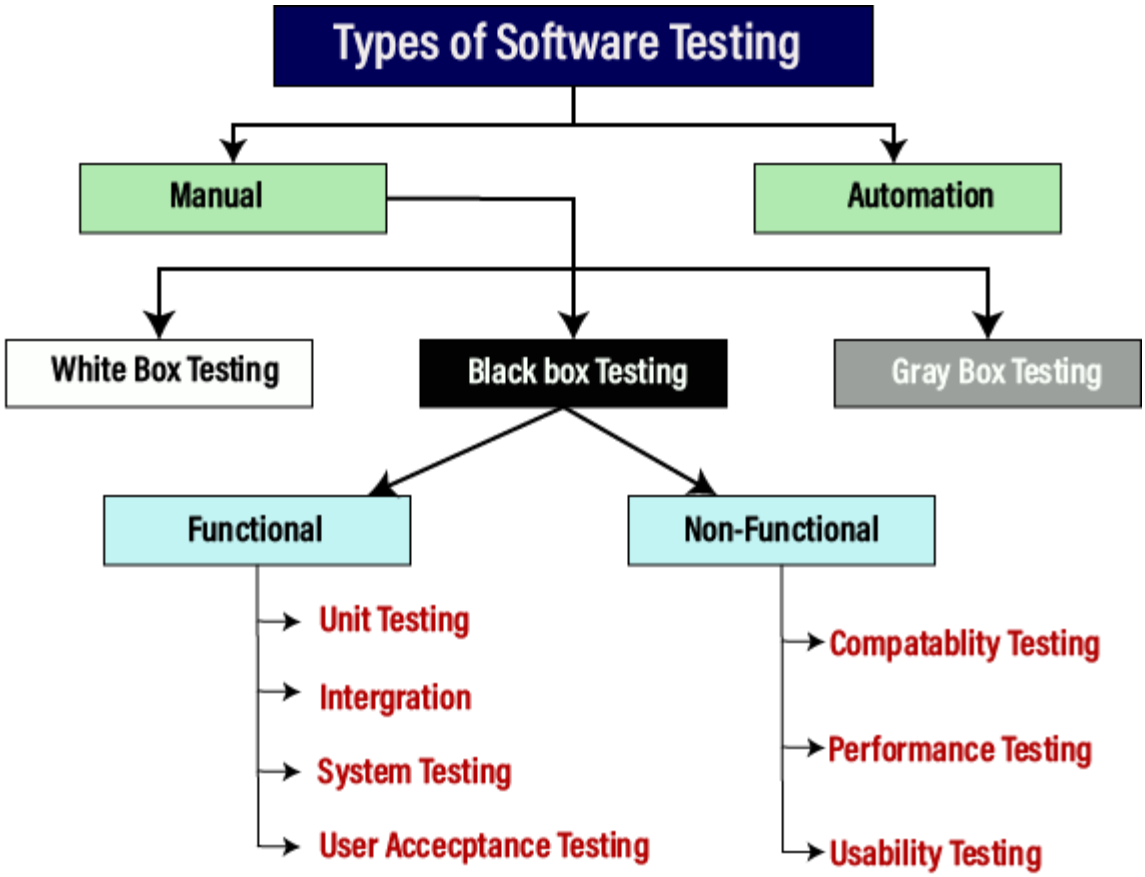
Types of software testing

Software Testing can be broadly classified into two types:

1. Manual Testing:
Manual testing includes testing software manually, i.e., without using any automation tool or any script. In this type, the tester takes over the role of an end-user and tests the software to identify any unexpected behavior or bug. There are different stages for manual testing such as unit testing, integration testing, system testing, and user acceptance testing.
Testers use test plans, test cases, or test scenarios to test software to ensure the completeness of testing. The software tester prepares test cases for different sections and levels of the code, executes the tests and reports the result to the manager.

Manual testing also includes exploratory testing, as testers explore the software to identify errors in it.

Manual testing is time and resource consuming. The tester needs to confirm whether or not right test cases are used. Major portion of testing involves manual testing.



2. Automation Testing: Automation testing, which is also known as Test Automation, is when the tester writes scripts and uses another software to test the product. This process involves the automation of a manual process. Automation Testing is used to re-run the test scenarios quickly and repeatedly, that were performed manually in manual testing.

Apart from regression testing, automation testing is also used to test the application from a load, performance, and stress point of view. It increases the test coverage, improves accuracy, and saves time and money when compared to manual testing.

Black box testing

It is also known as behavioral testing.

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

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.

The Black-Box can be any software system you want to test. For Example, an operating system like Windows, a website like Google, a database like Oracle or even your own custom application. Under Black Box Testing, you can test these applications by just focusing on the inputs and outputs without knowing their internal code implementation.

The primary source of black box testing is a specification of requirements that is stated by the customer.

In this method, 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. The test team reports the result to the development team and then tests the next function. After completing testing of all functions if there are severe problems, then it is given back to the development team for correction.

Types of Black Box Testing

There are many types of Black Box Testing but the following are the prominent ones –

- **Functional testing** – This black box testing type is related to the functional requirements of a system; it is done by software testers.
- **Non-functional testing** – This type of black box testing is not related to testing of specific functionality, but non-functional requirements such as performance, scalability, usability.
- **Regression testing** – Regression Testing is done after code fixes, upgrades or any other system maintenance to check the new code has not affected the existing code.

Tools used for Black Box Testing:

Tools used for Black box testing largely depends on the type of black box testing you are doing.

- For Functional/ Regression Tests you can use – QTP, Selenium
- For Non-Functional Tests, you can use – LoadRunner, Jmeter

Advantages of Black Box Testing:

- The tester does not need to have more functional knowledge or programming skills to implement the Black Box Testing.
- It is efficient for implementing the tests in the larger system.
- Tests are executed from the user's or client's point of view.
- Test cases are easily reproducible.
- It is used in finding the ambiguity and contradictions in the functional specifications.

Disadvantages of Black Box Testing:

- There is a possibility of repeating the same tests while implementing the testing process.
- Without clear functional specifications, test cases are difficult to implement.
- It is difficult to execute the test cases because of complex inputs at different stages of testing.
- Sometimes, the reason for the test failure cannot be detected.
- Some programs in the application are not tested.
- It does not reveal the errors in the control structure.

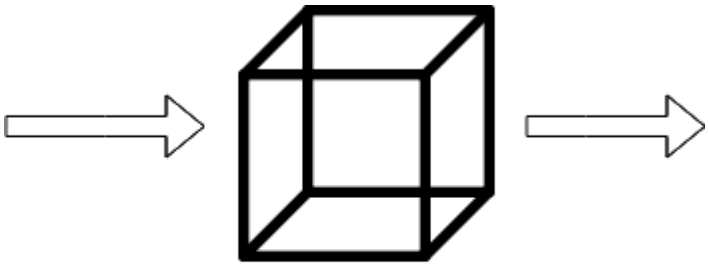
- Working with a large sample space of inputs can be exhaustive and consumes a lot of time.

White Box Testing

White box testing is also known as glass box testing, structural testing, clear box testing, open box testing and transparent box testing.

It tests internal coding and infrastructure of a software focus on checking of predefined inputs against expected and desired outputs. It is based on inner workings of an application and revolves around internal structure testing. In this type of testing programming skills are required to design test cases. The primary goal of white box testing is to focus on the flow of inputs and outputs through the software and strengthening the security of the software.

The term 'white box' is used because of the internal perspective of the system. The clear box or white box or transparent box name denote the ability to see through the software's outer shell into its inner workings.



Whitebox Testing

Developers do white box testing. In this, the developer will test every line of the code of the program. The developers perform the White-box testing and then send the application or the software to the testing team, where they will perform the black box testing and verify the application along with the requirements and identify the bugs and sends it to the developer.

The developer fixes the bugs and does one round of white box testing and sends it to the testing team. Here, fixing the bugs implies that the bug is deleted, and the particular feature is working fine on the application.

Here, the test engineers will not include in fixing the defects for the following reasons:

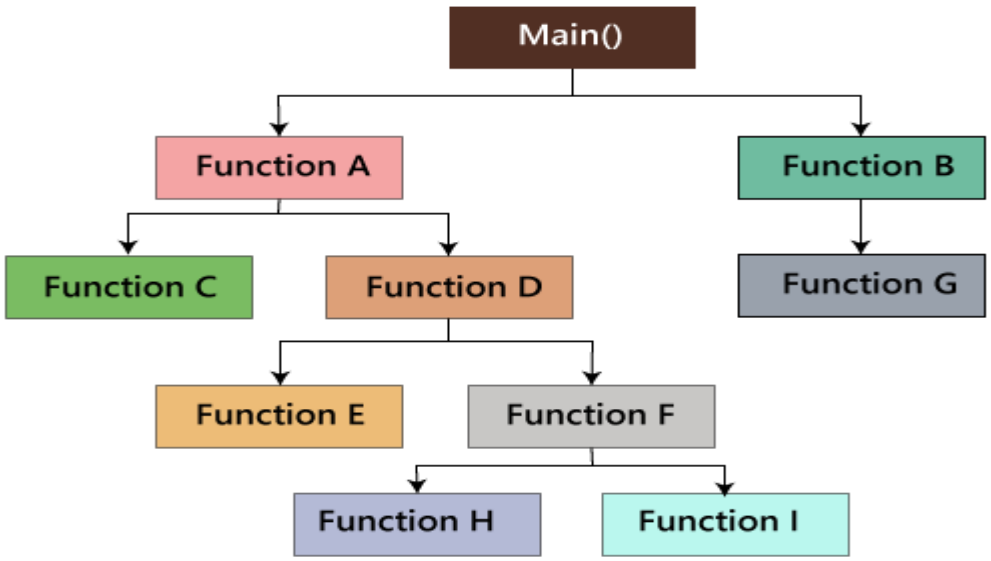
- *Fixing the bug might interrupt the other features. Therefore, the test engineer should always find the bugs, and developers should still be doing the bug fixes.*
- *If the test engineers spend most of the time fixing the defects, then they may be unable to find the other bugs in the application.*

The white box testing contains various tests, which are as follows:

- Path testing
- Loop testing
- Condition testing
- Testing based on the memory perspective
- Test performance of the program

Path testing

In the path testing, we will write the flow graphs and test all independent paths. Here writing the flow graph implies that flow graphs are representing the flow of the program and also show how every program is added with one another as we can see in the below image:



And test all the independent paths implies that suppose a path from main() to function G, first set the parameters and test if the program is correct in that particular path, and in the same way test all other paths and fix the bugs.

Loop testing

In the loop testing, we will test the loops such as while, for, and do-while, etc. and also check for ending condition if working correctly and if the size of the conditions is enough.

For example: we have one program where the developers have given about 50,000 loops.

```
1. {
2. while(50,000)
3. ……
4. ……
5. }
```

We cannot test this program manually for all the 50,000 loops cycle. So we write a small program that helps for all 50,000 cycles

Condition testing

In this, we will test all logical conditions for both true and false values; that is, we will verify for both if and else condition.

For example:

```
1.  if(condition) - true
2.  {
3.  …..
4.  ……
5.  ……
6.  }
7.  else - false
8.  {
9.  …..
10. ……
11. ……
12. }
```

The above program will work fine for both the conditions, which means that if the condition is accurate, and then else should be false and conversely.

Test cases for white box testing are derived from the design phase of the software development lifecycle. Data flow testing, control flow testing, path testing, branch testing, statement and decision coverage all these techniques used by white box testing as a guideline to create an error-free software.

White box testing follows some working steps to make testing manageable and easy to understand what the next task to do. There are some basic steps to perform white box testing.

Generic steps of white box testing

- Design all test scenarios, test cases and prioritize them according to high priority number.
- This step involves the study of code at runtime to examine the resource utilization, not accessed areas of the code, time taken by various methods and operations and so on.
- In this step testing of internal subroutines takes place. Internal subroutines such as nonpublic methods, interfaces are able to handle all types of data appropriately or not.
- This step focuses on testing of control statements like loops and conditional statements to check the efficiency and accuracy for different data inputs.
- In the last step white box testing includes security testing to check all possible security loopholes by looking at how the code handles security.

Reasons for white box testing

- It identifies internal security holes.
- To check the way of input inside the code.
- Check the functionality of conditional loops.
- To test function, object, and statement at an individual level.

Advantages of White box testing

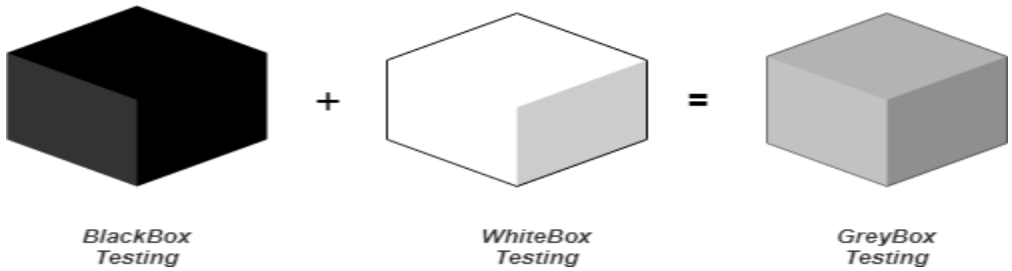
- White box testing optimizes code so hidden errors can be identified.
- Test cases of white box testing can be easily automated.
- This testing is more thorough than other testing approaches as it covers all code paths.
- It can be started in the SDLC phase even without GUI.

Disadvantages of White box testing

- White box testing is too much time consuming when it comes to large-scale programming applications.
- White box testing is much expensive and complex.
- It can lead to production error because it is not detailed by the developers.
- White box testing needs professional programmers who have a detailed knowledge and understanding of programming language and implementation.

GreyBox Testing

Greybox testing is a software testing method to test the software application with partial knowledge of the internal working structure. It is a **combination of black box and white box testing** because it involves access to internal coding to design test cases as white box testing and testing practices are done at functionality level as black box testing.

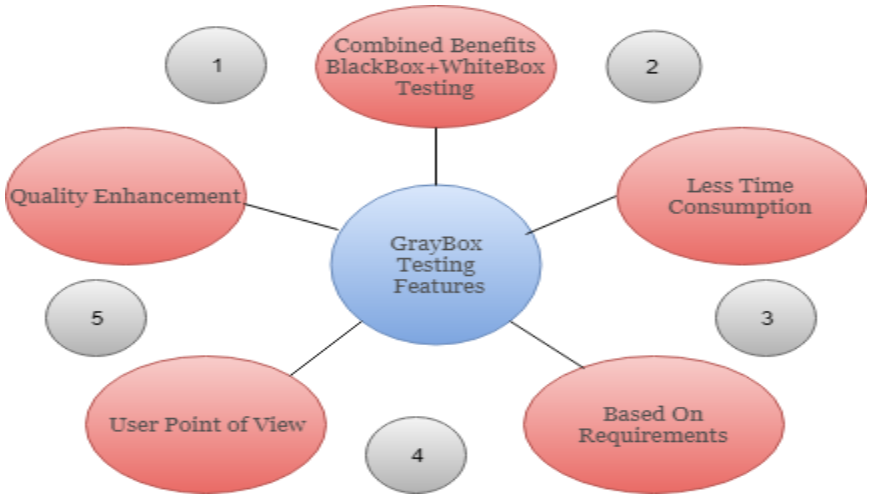


GreyBox testing commonly identifies context-specific errors that belong to web systems. For example; while testing, if tester encounters any defect then he makes changes in code to resolve the defect and then test it again in real time. It concentrates on all the layers of any complex software system to increase testing coverage. It gives the ability to test both presentation layer as well as internal coding structure. It is primarily used in integration testing and penetration testing.

Why GreyBox testing?

Reasons for GreyBox testing are as follows

- It provides combined benefits of both Blackbox testing and WhiteBox testing.
- It includes the input values of both developers and testers at the same time to improve the overall quality of the product.
- It reduces time consumption of long process of functional and non-functional testing.
- It gives sufficient time to the developer to fix the product defects.
- It includes user point of view rather than designer or tester point of view.
- It involves examination of requirements and determination of specifications by user point of view deeply.



S. No.	Black Box Testing	White Box Testing
1.	It is a way of software testing in which the internal structure or the program or the code is hidden and nothing is known about it.	It is a way of testing the software in which the tester has knowledge about the internal structure or the code or the program of the software.
2.	Implementation of code is not needed for black box testing.	Code implementation is necessary for white box testing.
3.	It is mostly done by software testers.	It is mostly done by software developers.
4.	No knowledge of implementation is needed.	Knowledge of implementation is required.
5.	It can be referred to as outer or external software testing.	It is the inner or the internal software testing.
6.	It is a functional test of the software.	It is a structural test of the software.
7.	This testing can be initiated based on the requirement specifications document.	This type of testing of software is started after a detail design document.
8.	No knowledge of programming is required.	It is mandatory to have knowledge of programming.
9.	It is the behavior testing of the software.	It is the logic testing of the software.
10.	It is applicable to the higher levels of testing of software.	It is generally applicable to the lower levels of software testing.
11.	It is also called closed testing.	It is also called as clear box testing.
12.	It is least time consuming.	It is most time consuming.
13.	It is not suitable or preferred for algorithm testing.	It is suitable for algorithm testing.
14.	Can be done by trial and error ways and methods.	Data domains along with inner or internal boundaries can be better tested.
15.	**Example:** Search something on google by using keywords	**Example:** By input to check and verify loops
16.	**Black-box test design techniques-** • Decision table testing • All-pairs testing • Equivalence partitioning • Error guessing	**White-box test design techniques-** • Control flow testing • Data flow testing • Branch testing
17.	**Types of Black Box Testing:** • Functional Testing • Non-functional testing • Regression Testing	**Types of White Box Testing:** • Path Testing • Loop Testing • Condition testing
18.	It is less exhaustive as compared to white box testing.	It is comparatively more exhaustive than black box testing.

(Type of Black Box Testing)

Functional Testing

Functional Testing is a type of Software Testing in which the system is tested against the functional requirements and specifications. Functional testing ensures that the requirements or specifications are properly satisfied by the application. This type of testing is particularly concerned with the result of processing.
The purpose of Functional tests is to test each function of the software application, by providing appropriate input, verifying the output against the Functional requirements.
Functional testing mainly involves black box testing and it is not concerned about the source code of the application. This testing checks User Interface, APIs, Database, Security, Client/Server communication and other functionality of the Application Under Test. The testing can be done either manually or using automation.
Functional testing can be manual or automated.

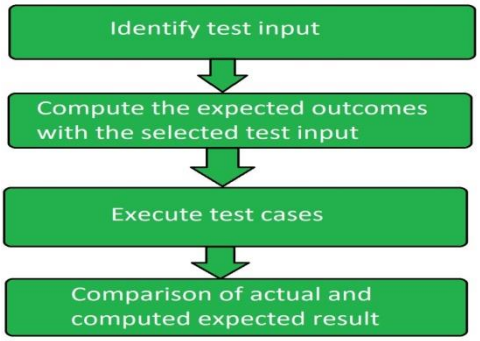
Goal of functional testing
The purpose of the functional testing is to check the primary entry function, necessarily usable function, the flow of screen GUI. Functional testing displays the error message so that the user can easily navigate throughout the application.

What do you test in Functional Testing?
The prime objective of Functional testing is checking the functionalities of the software system. It mainly concentrates on –

- **Mainline functions**: Testing the main functions of an application
- **Basic Usability**: It involves basic usability testing of the system. It checks whether a user can freely navigate through the screens without any difficulties.
- **Accessibility**: Checks the accessibility of the system for the user
- **Error Conditions**: Usage of testing techniques to check for error conditions. It checks whether suitable error messages are displayed.

Functional Testing Process:
Functional testing involves the following steps:

1. Identify function that is to be performed.
2. Create input data based on the specifications of function.
3. Determine the output based on the specifications of function.
4. Execute the test case.
5. Compare the actual and expected output.



Advantages of Functional Testing:

- It ensures to deliver a bug-free product.
- It ensures to deliver a high-quality product.
- No assumptions about the structure of the system.
- This testing is focused on the specifications as per the customer usage or it ensures that all requirements met.
- It ensures that the customer is satisfied.
- It ensures the proper working of all the functionality of an application/software/product.
- It ensures that the software/ product work as expected.
- It ensures security and safety.

Disadvantages of Functional Testing:

- There are high chances of performing redundant testing.
- If the requirement is not complete then performing this testing becomes difficult.
- Functional testing can miss a critical and logical error in the system.
- This testing is not a guarantee of the software to go live.

Major Functional Testing Techniques:

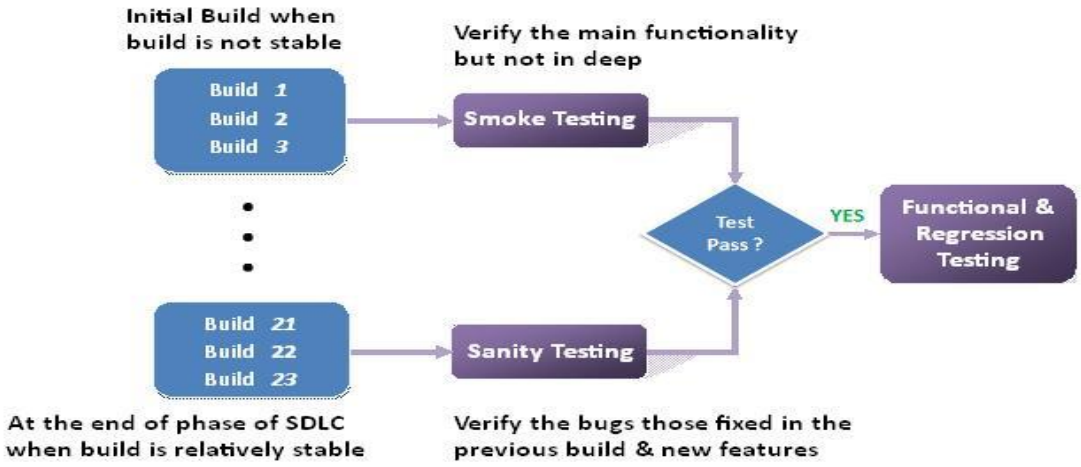
- Unit Testing
- Integration Testing
- Smoke Testing
- User Acceptance Testing
- Interface Testing
- System Testing
- Regression Testing

Smoke testing
When a new build is completed, it is handed to the Quality Assurance(QAs) for smoke testing. In this phase, only the most critical and core functionalities are tested to ensure that they yield the intended results. As an early-stage acceptance test, smoke testing adds a verification layer to determine whether or not the new build can proceed to the next stage or needs re-work.

Example: A utility company built an app with the function to report outages in customers' homes. This function reports the address and other relevant information as well as notifies the homeowner when a dispatcher is on the way to help. Smoke testing will validate this feature on a fundamental level to assure that when an outage is reported, the correct information is sent so a dispatcher can be there on time.

Regression testing
Any new change or feature added to the software can wreck its existing functionalities. Regression testing is performed every time alterations are made to check for the software's stability and functionalities. Due to its work-intensive nature, regression testing is often automated.

Example: A food delivery app added a function to help users add multiple promotions on top of each other. A regression test needs to be done to make sure the checkout and payment process is not affected.



Functional Testing Tools
Here is a list of popular **Functional Testing Tools**. They are explained as follows:

- **testRigor –** most advanced codeless UI end-to-end functional testing tool. Automate test cases in plain English, no matter how long or complex they are.
- **Selenium –** Popular Open Source Functional Testing Tool
- **QTP –** Very user-friendly Functional Test tool by HP
- **JUnit–** Used mainly for Java applications and this can be used in Unit and System Testing
- **soapUI –** This is an open source functional testing tool, mainly used for Web service testing. It supports multiple protocols such as HTTP, SOAP, and JDBC.
- **Watir –** This is a functional testing tool for web applications. It supports tests executed at the web browser and uses a ruby scripting language

Non Functional Testing

Non-functional Testing is a type of Software Testing that is performed to verify the non-functional requirements of the application. It verifies whether the behavior of the system is as per the requirement or not. It tests all the aspects which are not tested in functional testing. Non-Functional testing is a software testing technique that checks the non-functional attributes of the system. Non-functional testing is defined as a type of software testing to check non-functional aspects 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 as important as functional testing.

Objectives of Non-functional Testing

The objective of non-functional testing is:

- To increase usability, efficiency, maintainability and portability of the product.
- To help in the reduction of production risk related with non-functional aspects of the product.
- To help in the reduction of cost related with non-functional aspects of the product.
- To optimize the installation, execution and monitoring way of the product.
- To collect and produce measurements and metrics for internal research and development.
- To improve and enhance knowledge of the product behavior and technologies in use.

Non-Functional Testing Techniques

- **Compatibility testing:** A type of testing to ensure that a software program or system is compatible with other software programs or systems.
- **Compliance testing:** A type of testing to ensure that a software program or system meets a specific compliance standard, such as HIPAA or Sarbanes-Oxley.
- **Endurance testing:** A type of testing to ensure that a software program or system can handle a long-term, continuous load.
- **Load testing:** A type of testing to ensure that a software program or system can handle a large number of users or transactions.
- **Performance testing:** A type of testing to ensure that a software program or system meets specific performance goals, such as response time or throughput.
- **Recovery testing:** A type of testing to ensure that a software program or system can be recovered from a failure or data loss.
- **Security testing:** A type of testing to ensure that a software program or system is secure from unauthorized access or attack.
- **Scalability testing:** A type of testing to ensure that a software program or system can be scaled up or down to meet changing needs.
- **Stress testing:** A type of testing to ensure that a software program or system can handle an unusually high load.
- **Usability testing:** A type of testing to ensure that a software program or system is easy to use.
- **Volume testing:** A type of testing to ensure that a software program or system can handle a large volume of data.

Non-functional Testing Parameters

Techniques Used in Black Box Testing

Decision Table Technique	**Decision Table Technique is a systematic approach where various input combinations and their respective system behavior are captured in a tabular form. It is appropriate for the functions that have a logical relationship between two and more than two inputs.**
Boundary Value Technique	**Boundary Value Technique is used to test boundary values, boundary values are those that contain the upper and lower limit of a variable. It tests, while entering boundary value whether the software is producing correct output or not.**
State Transition	**State Transition Technique is used to capture the behavior of the software application when**

Technique	**different input values are given to the same function. This applies to those types of applications that provide the specific number of attempts to access the application.**
All-pair Testing Technique	**All-pair testing Technique 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, list box, text box, etc.**
Cause-Effect Technique	**Cause-Effect Technique underlines the relationship between a given result and all the factors affecting the result. It is based on a collection of requirements.**
Equivalence Partitioning Technique	**Equivalence partitioning is a technique of software testing in which input data divided into partitions of valid and invalid values, and it is mandatory that all partitions must exhibit the same behavior.**
Error Guessing Technique	**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.**
Use Case Technique	**Use case Technique 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.**

1. Decision table technique in Black box testing

Decision table technique is one of the widely used case design techniques for black box testing. This is a systematic approach where various input combinations and their respective system behavior are captured in a tabular form.

That's why it is also known as a cause-effect table. This technique is used to pick the test cases in a systematic manner; it saves the testing time and gives good coverage to the testing area of the software application.

Decision table technique is appropriate for the functions that have a logical relationship between two and more than two inputs.

This technique is related to the correct combination of inputs and determines the result of various combinations of input. To design the test cases by decision table technique, we need to consider conditions as input and actions as output.

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.

2. Boundary Value Analysis Black box Testing Technique

Boundary value analysis is one of the widely used case design technique for black box testing. It is used to test boundary values because the input values near the boundary have higher chances of error.

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

It is generally seen that a large number of errors occur at the boundaries of the defined input values rather than the center. It is also known as BVA and gives a selection of test cases which exercise bounding values.

This black box testing technique complements equivalence partitioning. This software testing technique base on the principle that, if a system works well for these particular values then it will work perfectly well for all values which comes between the two boundary values.

Guidelines for Boundary Value analysis

- If an input condition is restricted between values x and y, then the test cases should be designed with values x and y as well as values which are above and below x and y.
- If an input condition is a large number of values, the test case should be developed which need to exercise the minimum and maximum numbers. Here, values above and below the minimum and maximum values are also tested.
- Apply guidelines 1 and 2 to output conditions. It gives an output which reflects the minimum and the maximum values expected. It also tests the below or above values.

Example:

Input condition is valid between 1 to 10

Boundary values 0,1,2 and 9,10,11

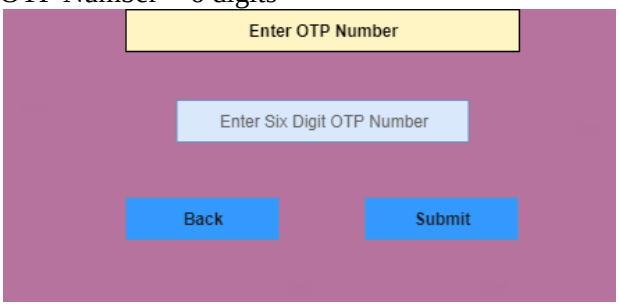
3. Equivalence Partitioning Technique

Equivalence partitioning is a technique of software testing in which input data is divided into partitions of valid and invalid values, and it is mandatory that all partitions must exhibit the same behavior. If a condition of one partition is true, then the condition of another equal partition must also be true, and if a condition of one partition is false, then the condition of another equal partition must also be false. 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.

Example:

Assume that there is a function of a software application that accepts a particular number of digits, not greater and less than that particular number. For example, an OTP number which contains only six digits, less or more than six digits will not be accepted, and the application will redirect the user to the error page.

- 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

We can see that there is a partition of two equally valid and invalid partitions, on applying valid value such as OTP of six digits in the example, both valid partitions behave same, i.e. redirected to the next page.
Another two partitions contain invalid values such as 5 or less than 5 and 7 or more than 7 digits in the example, and on applying these invalid values, both invalid partitions behave same, i.e. redirected to the error page.

We can see in the example, there are only three test cases for each example and that is also the principal of equivalence partitioning which states that this method intended to reduce the number of test cases.

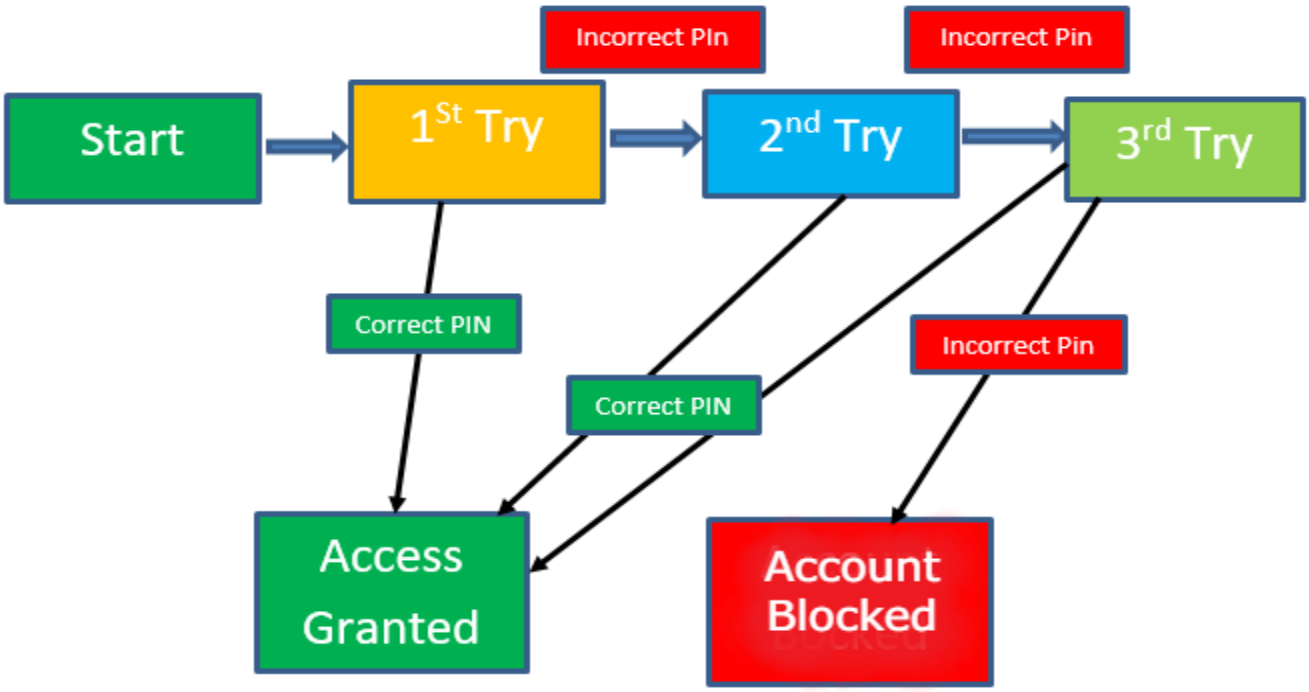
4. State Transition Technique of Black box Testing

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.

Example:

Let's consider an ATM system function where if the user enters the invalid password three times the account will be locked.
In this system, if the user enters a valid password in any of the first three attempts the user will be logged in successfully. If the user enters the invalid password in the first or second try, the user will be asked to re-enter the password. And finally, if the user enters incorrect password 3rd time, the account will be blocked.

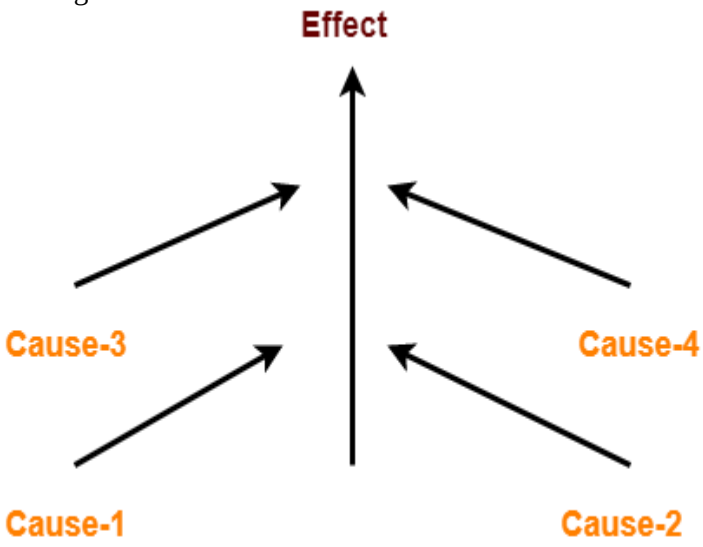
State Transition Diagram



This applies to those types of applications that provide the specific number of attempts to access the application such as the login function of an application which gets locked after the specified number of incorrect attempts.

5. Cause and Effect Graph Technique in Black box Testing

Cause Effect Graph is a popular black box testing technique.
It illustrates the relationship between a given outcome and all the factors that influence the outcome graphically



Cause-Effect Flow Diagram

Here,

- A “Cause” stands for a distinct input condition that fetches about an internal change in the system.
- An “Effect” represents an output condition, a system state that results from a combination of causes.

Applications-

- For analyzing the existing problem so that corrective actions can be taken at the earliest.
- For relating the interactions of the system with the factors affecting a particular process.
- For identifying the possible root causes, reasons for a particular effect, problem or outcome.

6. Error Guessing Technique in Black box Testing

Error Guessing is a software testing technique based on guessing the error which can prevail in the code. The technique is heavily based on the experience where the test analysts use their experience to guess the problematic part of the testing application. Hence, the test analysts must be skilled and experienced for better error guessing.
The technique counts a list of possible errors or error-prone situations. Then tester writes a test case to expose those errors. To design test cases based on this software testing technique, the analyst can use the past experiences to identify the conditions.

Guidelines for Error Guessing:

- The test should use the previous experience of testing similar applications
- Understanding of the system under test
- Knowledge of typical implementation errors
- Remember previously troubled areas
- Evaluate Historical data & Test results

Techniques Used in White Box Testing

Data Flow Testing	**Data flow testing is a group of testing strategies that examines the control flow of programs in order to explore the sequence of variables according to the sequence of events.**
Control Flow Testing	**Control flow testing determines the execution order of statements or instructions of the program through a control structure. The control structure of a program is used to develop a test case for the program. In this technique, a particular part of a large program is selected by the tester to set the testing path. Test cases represented by the control graph of the program.**
Branch Testing	**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.**
Statement Testing	**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.**
Decision Testing	**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.**

White Box Testing Technique

1. Data Flow Testing Technique in White box Testing

Data flow testing is used to analyze the flow of data in the program. It is the process of collecting information about how the variables flow the data in the program. It tries to obtain particular information of each particular point in the process.

Data flow testing is a group of testing strategies to examine the control flow of programs in order to explore the sequence of variables according to the sequence of events. It mainly focuses on the points at which values assigned to the variables and the point at which these values are used by concentrating on both points, data flow can be tested.

Data flow testing uses the control flow graph to detect illogical things that can interrupt the flow of data. Anomalies in the flow of data are detected at the time of associations between values and variables due to:

- If the variables are used without initialization.
- If the initialized variables are not used at least once.

Example,

```
1. read x;
2. If(x>0)
3. a= x+1
4. if (x<=0) {
5. if (x<1)
6. x=x+1; (go to 5)
else
7. a=x+1
8. print a;
```

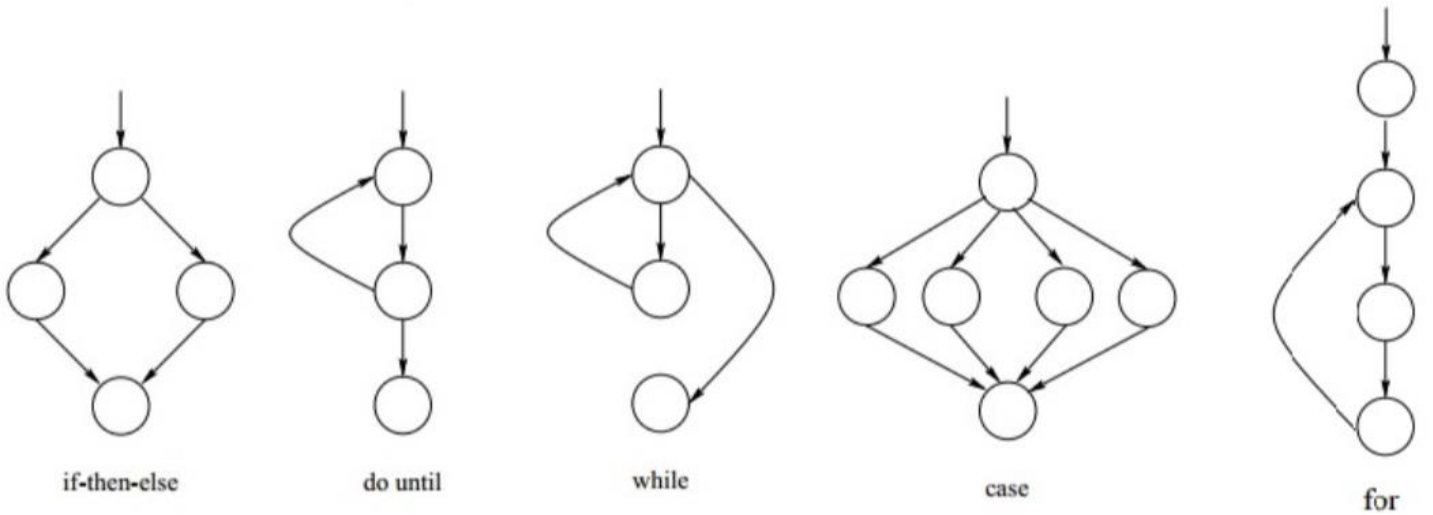
In this code, we have a total 8 statements, and we will choose a path which covers all the 8 statements. As it is evident in the code, we cannot cover all the statements in a single path because if statement 2 is true then statements 4, 5, 6, 7 not covered, and if statement 4 is true then statement 2 and 3 are not covered.

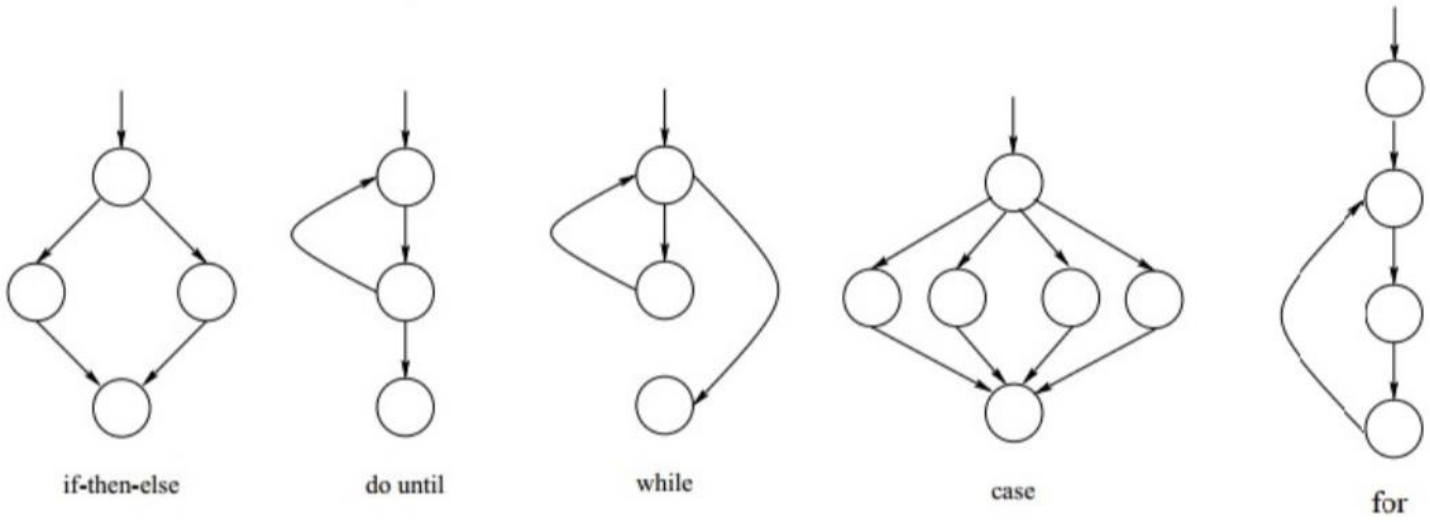
So, we are taking two paths to cover all the statements.
For the first path, we take the value of x is 1
1. x= 1
Path - 1, 2, 3, 8
For the second path, we take the value of x is -1
2. Set x= -1
Path = 1, 2, 4, 5, 6, 5, 6, 5, 7, 8

2. Control Flow Testing Technique in White box Testing

Control flow testing is a form of white-box testing where the implementation of the code is known to the tester. The development team often performs control flow testing. This process determines and observes the execution paths of a program in a structured way. We use this technique to develop the test cases.

We determine the different flows of the program before developing test cases for each case, which are then tested. This testing technique is mostly used in unit testing.
Some of the notations of program control flow are listed below:





Stages

1. **Control flow graph**

We can create the graph manually or with the help of software from the given source code.

2. **Coverage target**

Here, we include the nodes, edges, paths, branches, etc.

3. **Test case creation**

We create test cases with the help of control flow graphs whose purpose is to cover the defined coverage target.

4. **Test case execution**

We create and execute the test cases.

5. **Analysis of results**

In the end, we analyze the results to make sure that the system is error-free.

3. Branch Coverage Testing Technique in White box Testing

Branch Coverage is a white box testing method in which every outcome from a code module(statement or loop) is tested. The purpose of branch coverage is to ensure that each decision condition from every branch is executed at least once. It helps to measure fractions of independent code segments and to find out sections having no branches.

For example, if the outcomes are binary, you need to test both True and False outcomes.

Advantages of Branch coverage:

Branch coverage Testing offers the following advantages:

- Allows you to validate-all the branches in the code
- Helps you to ensure that no branched lead to any abnormality of the program's operation
- Branch coverage method removes issues which happen because of statement coverage testing
- Allows you to find those areas which are not tested by other testing methods
- It allows you to find a quantitative measure of code coverage
- Branch coverage ignores branches inside the Boolean expressions

4. Statement Coverage Testing Technique in White box Testing

Statement Coverage is a white box testing technique in which all the executable statements in the source code are executed at least once. It is used for calculation of the number of statements in source code which have been executed. The main purpose of Statement Coverage is to cover all the possible paths, lines and statements in source code.
Statement coverage is used to derive scenario based upon the structure of the code under test.

In White Box Testing, the tester is concentrating on how the software works. In other words, the tester will be concentrating on the internal working of source code concerning control flow graphs or flow charts.

Generally in any software, if we look at the source code, there will be a wide variety of elements like operators, functions, looping, exceptional handlers, etc. Based on the input to the program, some of the code statements may not be executed. The goal of Statement coverage is to cover all the possible path's, line, and statement in the code.

What is covered by Statement Coverage?

1. Unused Statements
2. Dead Code
3. Unused Branches
4. Missing Statements

5. Decision Coverage Testing Technique in White box Testing

Decision Coverage is a white box testing technique which reports the true or false outcomes of each boolean expression of the source code. The goal of decision coverage testing is to cover and validate all the accessible source code by checking and ensuring that each branch of every possible decision point is executed at least once.
In this coverage, expressions can sometimes get complicated. Therefore, it is very hard to achieve 100% coverage. That's why there are many different methods of reporting this metric. All these methods focus on covering the most important combinations. It is very much similar to decision coverage, but it offers better sensitivity to control flow.

Alpha Testing And Beta Testing

Alpha and Beta testing are Customer Validation methodologies (Acceptance Testing types) that help in building confidence to launch the product, and thereby result in the success of the product in the market.
Alpha and Beta Testing phases mainly focus on discovering the bugs from an already tested product and they give a clear picture of how the product is used by the real-time users. They also help in gaining experience with the product before its launch and valuable feedback is effectively implemented to increase the usability of the product.

Alpha Testing

Alpha testing is a type of acceptance testing, which is performed to identify all possible bugs/issues before releasing the product to the end-user. Alpha test is a preliminary software field test carried out by a team of users to find out the bugs that were not found previously by other tests. Alpha testing is to simulate a real user environment by carrying out tasks and operations that actual user might perform. Alpha testing implies a meeting with a software vendor and client to ensure that the developers appropriately meet the client's requirements in terms of the performance, functionality, and durability of the software.

Alpha testing needs lab environment, and usually, the testers are an internal employee of the organization mainly by the in-house software QA and testing teams. Alpha testing is the last testing done by the test teams at the development site after the acceptance testing and before releasing the software for beta test.

Alpha testing can also be done by potential users or customers of the application. Still, this is a form of in-house acceptance testing.

Beta Testing

This is a testing stage followed by the internal full alpha test cycle. This is the final testing phase where companies release the software to a few external user groups outside the company's test teams or employees. This initial software version is known as the beta version. Most companies gather user feedback in this release.
In short, beta testing can be defined as the testing carried out by real users in a real environment.
Though companies do rigorous in-house quality assurance from dedicated test teams, it's practically impossible to test an application for each and every combination of the test environment. Beta releases make it easier to test the application on thousands of test machines and fix the issues before releasing the application to the public.
The selection of beta test groups can be done based on the company's needs. The company can either invite a few users to test the preview version of the application or they can release it openly to give it a try by any user.
Fixing the issues in the beta release can significantly reduce the development cost as most of the minor glitches get fixed before the final release. So far, many big companies have successfully used beta versions of their most anticipated applications.
For example, recently Microsoft corporation released Windows 10 beta and based on the feedback from thousands of users they managed to release a stable OS version. In the past, Apple also released OS X beta in public and fixed many minor issues and improved the OS based on user feedback.

Alpha Testing Vs. Beta Testing

Alpha Testing	Beta Testing
Basic Understanding	
First phase of testing in Customer Validation	Second phase of testing in Customer Validation
Performed at developer's site - testing environment. Hence, the activities can be controlled	Performed in real environment, and hence activities cannot be controlled
Only functionality, usability are tested. Reliability and Security testing are not usually performed in-depth	Functionality, Usability, Reliability, Security testing are all given equal importance to be performed
White box and / or Black box testing techniques are involved	Only Black box testing techniques are involved
Build released for Alpha Testing is called Alpha Release	Build released for Beta Testing is called Beta Release
System Testing is performed before Alpha Testing	Alpha Testing is performed before Beta Testing
Issues / Bugs are logged into the identified tool directly and are fixed by developer at high priority	Issues / Bugs are collected from real users in the form of suggestions / feedbacks and are considered as improvements for future releases.
Helps to identify the different views of product usage as different business streams are involved	Helps to understand the possible success rate of the product based on real user's feedback / suggestions.
Test Goals	
To evaluate the quality of the product	To evaluate customer satisfaction
To ensure Beta readiness	To ensure Release readiness (for Production launch)
Focus on finding bugs	Focus on collecting suggestions / feedback and evaluate them effectively
Does the product work?	Do customers like the product?

Alpha Testing	Beta Testing
When	
Usually after System testing phase or when the product is 70% - 90% complete	Usually after Alpha Testing and product is 90% - 95% complete
Features are almost freezed and no scope for major enhancements	Features are freezed and no enhancements accepted
Build should be stable for technical user	Build should be stable for real users
Test Duration	
Many test cycles conducted	Only 1 or 2 test cycles conducted
Each test cycle lasts for 1 - 2 weeks	Each test cycle lasts for 4 - 6 weeks
Duration also depends on the number of issues found and number of new features added	Test cycles may increase based on real user's feedback / suggestion
Stake Holders	
Engineers (in-house developers), Quality Assurance Team, and Product Management Team	Product Management, Quality Management, and User Experience teams
Participants	
Technical Experts, Specialized Testers with good domain knowledge (new or who were already part of System Testing phase), Subject Matter Expertise	End users to whom the product is designed
Customers and / or End Users can participate in Alpha Testing in some cases	Customers also usually participate in Beta Testing
Expectations	
Acceptable number of bugs that were missed in earlier testing activities	Major completed product with very less amount of bugs and crashes
Incomplete features and documentation	Almost completed features and documentation
Entry Criteria	
• Alpha Tests designed and reviewed for Business requirements • Traceability matrix should be achieved for all the between alpha tests and requirements • Testing team with knowledge about the domain and product • Environment setup and build for execution • Tool set up should be ready for bug logging and test management System testing should be signed-off (ideally)	• Beta Tests like what to test and procedures documented for Product usage • No need of Traceability matrix • Identified end users and customer team up • End user environment setup • Tool set up should be ready to capture the feedback / suggestions • Alpha Testing should be signed off
Exit Criteria	
• All the alpha tests should be executed and all the cycles should be completed • Critical / Major issues should be fixed and retested • Effective review of feedback provided by participants should be completed • Alpha Test Summary report • Alpha testing should be signed off	• All the cycles should be completed • Critical / Major issues should be fixed and retested • Effective review of feedback provided by participants should be completed • Beta Test summary report • Beta Testing should be signed off
Rewards	
No specific rewards or prizes for participants	Participants are rewarded
Pros	
• Helps to uncover bugs that were not found during previous testing activities • Better view of product usage and reliability • Analyze possible risks during and after launch of the product • Helps to be prepared for future customer support • Helps to build customer faith on the product • Maintenance Cost reduction as the bugs are identified and fixed before Beta / Production launch	• Product testing is not controllable and user may test any available feature in any way - corner areas are well tested in this case • Helps to uncover bugs that were not found during previous testing activities (including alpha) • Better view of product usage, reliability, and security • Analyze the real user's perspective and opinion on the product • Feedback / suggestions from real users helps in improvising the product in future

Alpha Testing	Beta Testing
• Easy Test Management	• Helps to increase customer satisfaction on the product
Cons	
• Not all the functionality of the product is expected to be tested • Only Business requirements are scoped	• Scope defined may or may not be followed by participants • Documentation is more and time consuming - required for using bug logging tool (if required), using tool to collect feedback / suggestion, test procedure (installation / uninstallation, user guides) • Not all the participants assure to give quality testing • Not all the feedback are effective - time taken to review feedback is high • Test Management is too difficult
What Next	
Beta Testing	Field Testing

Unit Testing

Unit Testing is a software testing technique by means of which individual units of software i.e. group of computer program modules, usage procedures, and operating procedures are tested to determine whether they are suitable for use or not.
It is a testing method using which every independent module is tested to determine if there is an issue by the developer himself. It is correlated with the functional correctness of the independent modules.
Unit Testing is defined as a type of software testing where individual components of a software are tested. Unit Testing of the software product is carried out during the development of an application. An individual component may be either an individual function or a procedure. Unit Testing is typically performed by the developer.
In SDLC or V Model, Unit testing is the first level of testing done before integration testing. Unit testing is such a type of testing technique that is usually performed by developers. Although due to the reluctance of developers to test, quality assurance engineers also do unit testing.

Why perform Unit Testing?
Unit Testing is important because software developers sometimes try saving time doing minimal unit testing and this is myth because inappropriate unit testing leads to high cost Defect fixing during System Testing, Integration Testing and even Beta Testing after application is built. If proper unit testing is done in early development, then it saves time and money in the end.
Some crucial reasons are listed below:

- Unit testing helps tester and developers to understand the base of code that makes them able to change defect causing code quickly.
- Unit testing helps in the documentation.
- 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.
- It helps with code reusability by migrating code and test cases.

Objective of Unit Testing:
The objective of Unit Testing is:

- To isolate a section of code.
- To verify the correctness of the code.
- To test every function and procedure.
- To fix bugs early in the development cycle and to save costs.
- To help the developers to understand the code base and enable them to make changes quickly.
- To help with code reuse.

Types of Unit Testing:
There are 2 types of Unit Testing: Manual, and Automated.

Unit testing is commonly automated but may still be performed manually. Software Engineering does not favor one over the other but automation is preferred. A manual approach to unit testing may employ a step-by-step instructional document.

Unit Testing Techniques:
There are 3 types of Unit Testing Techniques. They are

- **Black Box Testing:** This testing technique is used in covering the unit tests for input, user interface, and output parts.
- **White Box Testing:** This technique is used in testing the functional behavior of the system by giving the input and checking the functionality output including the internal design structure and code of the modules.
- **Gray Box Testing:** This technique is used in executing the relevant test cases, test methods, test functions, and analyzing the code performance for the modules.

(Unit testing uses all white box testing techniques as it uses the code of software application:)

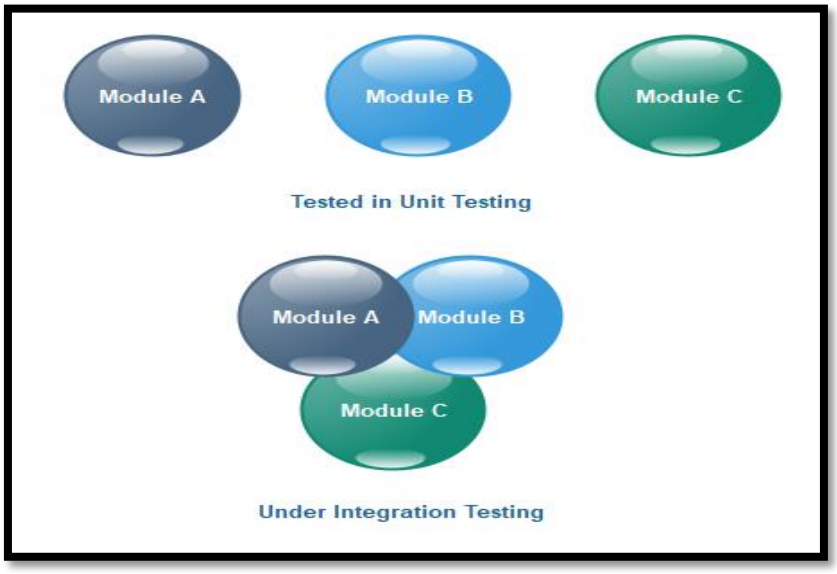
- Data flow Testing
- Control Flow Testing
- Branch Coverage Testing
- Statement Coverage Testing
- Decision Coverage Testing

Advantages of Unit Testing:

- Unit Testing allows developers to learn what functionality is provided by a unit and how to use it to gain a basic understanding of the unit API.
- Unit testing allows the programmer to refine code and make sure the module works properly.
- Unit testing enables testing parts of the project without waiting for others to be completed.

Disadvantages of Unit Testing:

- The process is time-consuming for writing the unit test cases.
- Unit Testing will not cover all the errors in the module because there is a chance of having errors in the modules while doing integration testing.
- Unit Testing is not efficient for checking the errors in the UI(User Interface) part of the module.
- It requires more time for maintenance when the source code is changed frequently.
- It cannot cover the non-functional testing parameters such as scalability, the performance of the system, etc.



Integration Testing

Integration testing is the process of testing the interface between two software units or modules. It focuses on determining the correctness of the interface. The purpose of integration testing is to expose faults in the interaction between integrated units. Once all the modules have been unit tested, integration testing is performed.

Integration testing can be done by picking module by module. This can be done so that there should be proper sequence to be followed. And also if you don't want to miss out on any integration scenarios then you have to follow the proper sequence. Exposing the defects is the major focus of the integration testing and the time of interaction between the integrated units.
Integration Testing focuses on checking data communication amongst these modules. Hence it is also termed as 'I & T' (Integration and Testing), 'String Testing' and sometimes 'Thread Testing'.

Why do Integration Testing?

Although each software module is unit tested, defects still exist for various reasons like

- A Module, in general, is designed by an individual software developer whose understanding and programming logic may differ from other programmers. Integration Testing becomes necessary to verify the software modules work in unity
- At the time of module development, there are wide chances of change in requirements by the clients. These new requirements may not be unit tested and hence system integration Testing becomes necessary.
- Interfaces of the software modules with the database could be erroneous
- External Hardware interfaces, if any, could be erroneous
- Inadequate exception handling could cause issues.

How to do Integration Testing?

The Integration test procedure irrespective of the Software testing strategies (discussed above):

- Prepare the Integration Tests Plan
- Design the Test Scenarios, Cases, and Scripts.
- Executing the test Cases followed by reporting the defects.
- Tracking & re-testing the defects.
- Steps 3 and 4 are repeated until the completion of Integration is successful.

Integration test approaches

There are four types of integration testing approaches. Those approaches are the following:

1. Big-Bang Integration Testing –

It is the simplest integration testing approach, where all the modules are combined and the functionality is verified after the completion of individual module testing. In simple words, all the modules of the system are simply put together and tested. This approach is practicable only for very small systems. If an error is found during the integration testing, it is very difficult to localize the error as the error may potentially belong to any of the modules being integrated. So, debugging errors reported during big bang integration testing is very expensive to fix.

Advantages:

- It is convenient for small systems.

Disadvantages:

- There will be quite a lot of delay because you would have to wait for all the modules to be integrated.
- High risk critical modules are not isolated and tested on priority since all modules are tested at once.
- Not Good for long Projects.

2. Bottom-Up Integration Testing –

In bottom-up testing, each module at lower levels is tested with higher modules until all modules are tested. The primary purpose of this integration testing is that each subsystem tests the interfaces among various modules making up the subsystem. This integration testing uses test drivers to drive and pass appropriate data to the lower level modules.

Advantages:

- In bottom-up testing, no stubs are required.
- A principle advantage of this integration testing is that several disjoint subsystems can be tested simultaneously.
- It is easy to create the test conditions.
- Best for the applications that uses bottom up design approach.
- It is Easy to observe the test results.

Disadvantages:

- Driver modules must be produced.
- In this testing, the complexity that occurs when the system is made up of a large number of small subsystems.
- As Far modules have been created, there is no working model can be represented.

3. Top-Down Integration Testing –

Top-down integration testing technique is used in order to simulate the behaviour of the lower-level modules that are not yet integrated. In this integration testing, testing takes place from top to bottom. First, high-level modules are tested and then low-level modules and finally integrating the low-level modules to a high level to ensure the system is working as intended.

Advantages:

- Separately debugged module.
- Few or no drivers needed.
- It is more stable and accurate at the aggregate level.
- Easier isolation of interface errors.
- In this, design defects can be found in the early stages.

Disadvantages:

- Needs many Stubs.
- Modules at lower level are tested inadequately.
- It is difficult to observe the test output.
- It is difficult to stub design.

4. Mixed Integration Testing – (Sandwich Testing/Hybrid Integration Testing)

A mixed integration testing is also called sandwiched integration testing. A mixed integration testing follows a combination of top down and bottom-up testing approaches. In top-down approach, testing can start only after the top-level module have been coded and unit tested. In bottom-up approach, testing can start only after the bottom level modules are ready. This sandwich or mixed approach overcomes this shortcoming of the top-down and bottom-up approaches. It is also called the hybrid integration testing. also, stubs and drivers are used in mixed integration testing.

Advantages:

- Mixed approach is useful for very large projects having several sub projects.
- This Sandwich approach overcomes this shortcoming of the top-down and bottom-up approaches.
- Parallel test can be performed in top and bottom layer tests.

Disadvantages:

- For mixed integration testing, it requires very high cost because one part has Top-down approach while another part has bottom-up approach.
- This integration testing cannot be used for smaller systems with huge interdependence between different modules.

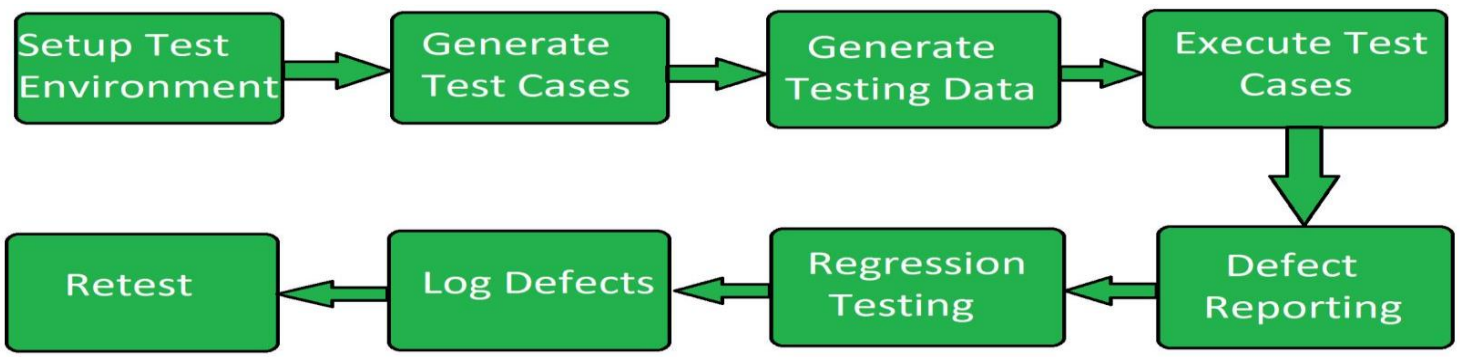
System Testing

System testing is a type of software testing that is performed on a complete integrated system to evaluate the compliance of the system with the corresponding requirements. In system testing, integration testing passed components are taken as input. The goal of integration testing is to detect any irregularity between the units that are integrated together. System testing detects defects within both the integrated units and the whole system. The result of system testing is the observed behavior of a component or a system when it is tested. **System Testing** is carried out on the whole system in the context of either system requirement specifications or functional requirement specifications or in the context of both. System testing tests the design and behavior of the system and also the expectations of the customer. It is performed to test the system beyond the bounds mentioned in the software requirements specification (SRS). System Testing is basically performed by a testing team that is independent of the development team that helps to test the quality of the system impartial. It has both functional and non-functional testing. **System Testing is a black-box testing**. System Testing is performed after the integration testing and before the acceptance testing.

System Testing Process:

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.



Types of System Testing:

- **Performance Testing:** Performance Testing is a type of software testing that is carried out to test the speed, scalability, stability and reliability of the software product or application.
- **Load Testing:** Load Testing is a type of software Testing which is carried out to determine the behavior of a system or software product under extreme load.
- **Stress Testing:** Stress Testing is a type of software testing performed to check the robustness of the system under the varying loads.
- **Scalability Testing:** Scalability Testing is a type of software testing which is carried out to check the performance of a software application or system in terms of its capability to scale up or scale down the number of user request load.

Advantages of System Testing:

- The testers do not require more knowledge of programming to carry out this testing.
- It will test the entire product or software so that we will easily detect the errors or defects which cannot be identified during the unit testing and integration testing.
- The testing environment is similar to that of the real time production or business environment.
- It checks the entire functionality of the system with different test scripts and also it covers the technical and business requirements of clients.
- After this testing, the product will almost cover all the possible bugs or errors and hence the development team will confidently go ahead with acceptance testing.

Disadvantages of System Testing:

- This testing is time consuming process than another testing techniques since it checks the entire product or software.
- The cost for the testing will be high since it covers the testing of entire software.
- It needs good debugging tool otherwise the hidden errors will not be found.

Debugging

In the context of software engineering, debugging is the process of fixing a bug in the software. In other words, it refers to identifying, analyzing, and removing errors. This activity begins after the software fails to execute properly and concludes by solving the problem and successfully testing the software. It is considered to be an extremely complex and tedious task because errors need to be resolved at all stages of debugging.

Debugging Process:

Steps involved in debugging are:

- Problem identification and report preparation.
- Assigning the report to the software engineer to the defect to verify that it is genuine.
- Defect Analysis using modeling, documentation, finding and testing candidate flaws, etc.
- Defect Resolution by making required changes to the system.
- Validation of corrections.

The debugging process will always have one of two outcomes:

1. The cause will be found and corrected.
2. The cause will not be found.

Later, the person performing debugging may suspect a cause, design a test case to help validate that suspicion and work toward error correction in an iterative fashion.

During debugging, we encounter errors that range from mildly annoying to catastrophic. As the consequences of an error increase, the amount of pressure to find the cause also increases. Often, pressure sometimes forces a software developer to fix one error and at the same time introduce two more.

Debugging Approaches/Strategies:

1. **Brute Force:** Study the system for a larger duration in order to understand the system. It helps the debugger to construct different representations of systems to be debugging depending on the need. A study of the system is also done actively to find recent changes made to the software.
2. **Backtracking:** Backward analysis of the problem which involves tracing the program backward from the location of the failure message in order to identify the region of faulty code. A detailed study of the region is conducted to find the cause of defects.
3. **Forward analysis** of the program involves tracing the program forwards using breakpoints or print statements at different points in the program and studying the results. The region where the wrong outputs are obtained is the region that needs to be focused on to find the defect.
4. **Using the past experience** of the software debug the software with similar problems in nature. The success of this approach depends on the expertise of the debugger.
5. **Cause elimination:** it introduces the concept of binary partitioning. Data related to the error occurrence are organized to isolate potential causes.

Difference Between Debugging and Testing:
Debugging is different from testing. Testing focuses on finding bugs, errors, etc whereas debugging starts after a bug has been identified in the software. Testing is used to ensure that the program is correct and it was supposed to do with a certain minimum success rate. Testing can be manual or automated. There are several different types of testing like unit testing, integration testing, alpha and beta testing, etc. Debugging requires a lot of knowledge, skills, and expertise. It can be supported by some automated tools available but is more of a manual process as every bug is different and requires a different technique, unlike a pre-defined testing mechanism.

Software Testing Strategies

Software testing strategy is an approach which incorporates planning of the steps to test the software along with the planning of time, effort & resources that will be required to test the software. Software testing strategies plans to test any software starting from the smallest component of the software and then integrating test towards the complete system.

Software testing strategy is the planning done before testing commences and exercised systematically to test the software. The testing strategy could be developed by the project manager, or by the software engineers or it could even be a testing specialist.

Developing a testing strategy for software is important because if testing is not conducted properly it would lead to wastage of time and effort and it would even be the case that some error or bugs remain undetected. Some general **characteristics** that should be considered while developing the testing strategy are as follow:
1. For successful testing, you should conduct the technical evaluation, it would reveal many of the error before the testing starts. It would help in correcting the technical error before the testing commences & would save time while testing the software.

2. Testing must start from the core of the software design and it must progress outward to incorporate testing of the entire software.

3. You should not follow the same strategy to test all the software. Appropriate testing strategies must be developed for different software engineering approach.

4. Generally, the developer of the software conduct the software testing, and in case the project is large a separate team must be allotted to test the software.

5. Though debugging and testing are two distinct jobs; debugging must be incorporated in every testing strategy.

The **software testing strategy** is implemented at two levels**, low-level testing** and **high-level testing**. Low-level testing is conducted at the core of software design like **verifying** the code segments of the software. Whereas, high-level testing is conducted to **validate** software functions against the specified requirements of the customers.

Verification and Validation

Verifying and validating a software incorporates a wide range of software quality assurance activities. By **verifying** a software, we ensure that every function in the software is implemented and functioning accurately. By **validating** software, we ensure that the developed software meets the requirements specified by the customer.

In a simple language, **verification** answers the question of whether 'we are implementing the software correctly?' & **validating** answers, the question of whether 'we are implementing the correct software?'.

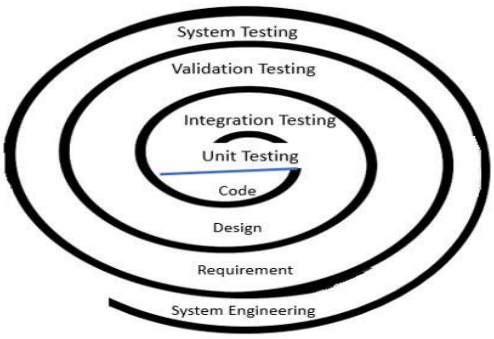
Levels of Software Testing Strategies

While developing the software the software engineers first review the customer requirements, then models it and finally go for coding. That means while developing the software developers move inwards the spiral in an anti-clockwise direction.

For testing, initially, the core part of the software is tested, each unit of the software is tested separately to ensure that as a unit this component functions properly. Then the testing proceeds outward the spiral along the streamlines. Moving outwards the spiral in the clockwise direction gradually widens the scope of testing and in each turn, testing integrates the components and ensure that all the elements of the software are functioning properly and the desired performance is achieved.

Levels of testing strategies that are developed to test the entire software are:

(explain levels of software testig)



Software Testing Strategy