Snehal Thomake

Date=15/10/2022

Software Testing

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.

**Software testing can be divided into two steps:**

1. **Verification:** it refers to the set of tasks that ensure that the software correctly implements a specific function.

2. **Validation:** it refers to a different set of tasks that ensure that the software that has been built is traceable to customer requirements.

## Software Bugs

A software bug is a flaw, failure, error or fault in a computer software or system that causes it to return unexpected or incorrect results.

A software bug is an error, flaw, or fault in an application. This error causes the application to produce an unintended or unexpected result, such as crashing or producing invalid results.

Software bugs can be caused by many factors, including unclear requirements, programming errors, software complexity, lack of communication, timeline deviation, errors in bug tracking, documentation errors, deviation from standards, and much more.

**Mistakes**

Optimize the decisions that define your code by exploring the common mistakes and intentional tradeoffs made by expert developers.

**Error**

An Error is a mistake made in the code; that's why we cannot execute or compile code. The Fault is a state that causes the software to fail to accomplish its essential function. If the software has lots of defects, it leads to failure or causes failure.

**Faults**

Software fault is also known as defect, arises when the expected result don't match with the actual results. It can also be error, flaw, failure, or fault in a computer program. Most bugs arise from mistakes and errors made by developers, architects.

**Defect**

A Bug is the result of a coding Error and A Defect is a deviation from the Requirements. A defect does not necessarily mean there is a bug in the code, it could be a function that was not implemented but defined in the requirements of the software

**Failure**

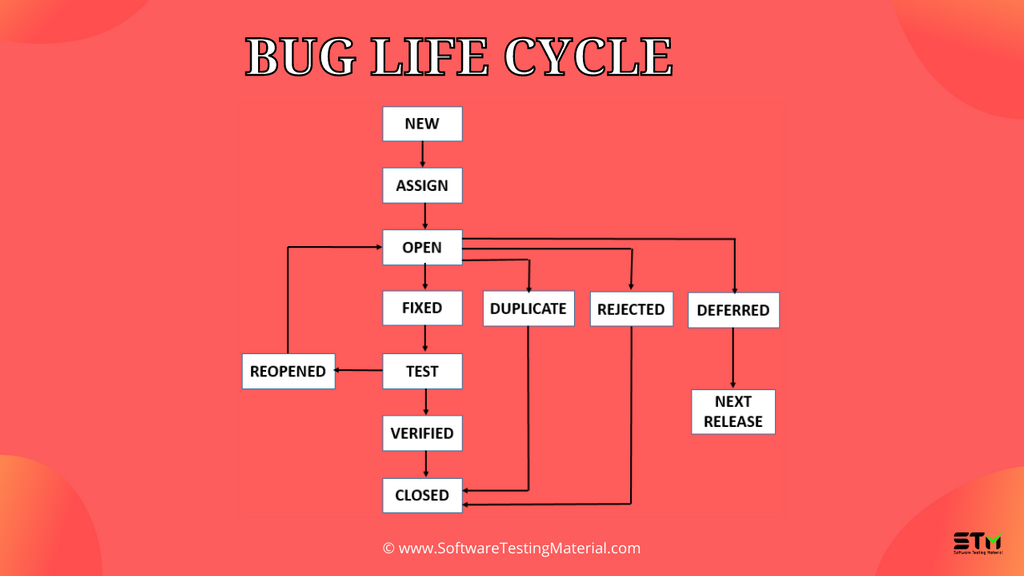
A failure that occurs when the user perceives that the software has ceased to deliver the expected result with respect to the specification input values. The user may need to identify the severity of the levels of failures such as catastrophic, critical, major or minor, depending on their impact on the systems.

## What is Defect/Bug?

A **defect** is an **error** or a **bug**, in the application which is created. A programmer while designing and building the software can make **mistakes** or **errors**. These mistakes or errors mean that there are **flaws** in the software. These are called **defects**.

## What is Defect Life Cycle?

**Defect life cycle**, also known as **Bug Life cycle** is the journey of a defect cycle, which a defect goes through during its lifetime. It varies from organization to organization and also from project to project as it is governed by the software testing process and also depends upon the tools used.



### ****Defect Life Cycle includes the following stages:****

**New:** When a defect is logged and posted for the first time. Its state is given as new.

**Assigned:** Once the bug is posted by the tester, the lead of the tester approves the bug and assigns the bug to the developer team. There can be two scenarios, first that the defect can directly assign to the developer, who owns the functionality of the defect. Second, it can also be assigned to the Dev Lead and once it is approved with the Dev Lead, he or she can further move the defect to the developer.

**Open:** Its state when the developer starts analyzing and working on the defect fix.

**Fixed:** When developer makes necessary code changes and verifies the changes then he/she can make bug status as ‘Fixed’. This is also an indication to the Dev Lead that the defects on Fixed status are the defect which will be available to tester to test in the coming build.

**Retest:** At this stage the tester do the retesting of the changed code which developer has given to him to check whether the defect got fixed or not.

Once the latest build is pushed to the environment, Dev lead move all the Fixed defects to Retest. It is an indication to the testing team that the defects are ready to test.

**Reopened**:  If the bug still exists even after the bug is fixed by the developer, the tester changes the status to “**reopened**”. The bug goes through the life cycle once again.

**Deferred**: The bug, changed to deferred state means the bug is expected to be fixed in next releases. The reasons for changing the bug to this state have many factors. Some of them are priority of the bug may be low, lack of time for the release or the bug may not have major effect on the software.

**Rejected**: If the developer feels that the bug is not genuine, developer rejects the bug. Then the state of the bug is changed to “**rejected**”.

**Duplicate** : If the bug is repeated twice or the two bugs mention the same concept of the bug, then the recent/latest bug status is changed to “**duplicate**“.

**Closed**:  Once the bug is fixed, it is tested by the tester. If the tester feels that the bug no longer exists in the software, tester changes the status of the bug to “**closed**”. This state means that the bug is fixed, tested and approved.

**Not a bug/Enhancement**:  The state given as “**Not a bug/Enhancement**” if there is no change in the functionality of the application. For an example: If customer asks for some change in the look and field of the application like change of color of some text then it is not a bug but just some change in the looks of the  application.

# Black box testing

**Black box** testing is a type of software testing in which the functionality of the software is not known. The testing is done without the internal knowledge of the products.

# White box Testing

White box testing techniques analyze the internal structures the used data structures, internal design, code structure and the working of the software rather than just the functionality as in black box testing. It is also called glass box testing or clear box testing or structural testing.

|  |  |
| --- | --- |
| Black Box Testing | White Box Testing |
| It is a testing method without having knowledge about the actual code or internal structure of the application. | **It is a testing method having knowledge about the actual code and internal structure of the application.** |
| This is a higher level testing such as functional testing. | **This type of testing is performed at a lower level of testing such as Unit Testing, Integration Testing.** |
| It concentrates on the functionality of the system under test. | **It concentrates on the actual code – program and its syntax's.** |
| Black box testing requires Requirement specification to test. | **White Box testing requires Design documents with data flow diagrams, flowcharts etc.** |
| Black box testing is done by the testers. | **White box testing is done by Developers or testers with programming knowledge.** |

**There are seven principles in software testing:** 

1. Testing shows the presence of defects
2. Exhaustive testing is not possible
3. Early testing
4. Defect clustering
5. Pesticide paradox
6. Testing is context-dependent
7. Absence of errors fallacy



### What is Black Box Testing?

Black Box Testing is also known as behavioral, opaque-box, closed-box, specification-based or eye-to-eye testing.

It is a Software Testing method that analyzes the functionality of a software/application without knowing much about the internal structure/design of the item that is being tested and compares the input value with the output value.

### Types of Black Box Testing

Practically, there are several types of Black Box Testing that are possible, but if we consider a major variant of it then only the below mentioned are the two fundamental ones.

#### #1) Functional Testing

This testing type deals with the functional requirements or specifications of an application. Here, different actions or functions of the system are being tested by providing the input and comparing the actual output with the expected output.

**For example**, when we test a Dropdown list, we click on it and verify if it expands and all the expected values are showing in the list.

**Few major types of Functional Testing are:**

* Smoke Testing
* Sanity Testing
* Integration Testing
* System Testing
* Regression Testing
* User Acceptance Testing

#### #2) Non-Functional Testing

Apart from the functionalities of the requirements, there are even several non-functional aspects that are required to be tested to improve the quality and performance of the application.

**Few major types of Non-Functional Testing include:**

* Usability Testing
* Load Testing
* Performance Testing
* Compatibility Testing
* Stress Testing
* Scalability Testing

### Black Box Testing Techniques

In order to systematically test a set of functions, it is necessary to design test cases. Testers can create test cases from the requirement specification document using the following Black Box Testing techniques:

* Equivalence Partitioning
* Boundary Value Analysis
* Decision Table Testing
* State Transition Testing
* Error Guessing
* Graph-Based Testing Methods
* Comparison Testing

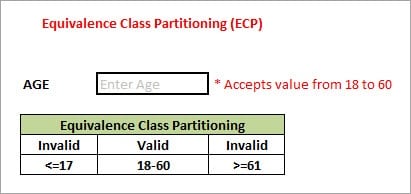
***Let’s understand each technique in detail.***

#### #1) Equivalence Partitioning

This technique is also known as Equivalence Class Partitioning (ECP). In this technique, input values to the system or application are divided into different classes or groups based on its similarity in the outcome.

Hence, instead of using each and every input value, we can now use any one value from the group/class to test the outcome. This way, we can maintain test coverage while we can reduce the amount of rework and most importantly the time spent.

**For Example:**



As present in the above image, the “AGE” text field accepts only numbers from 18 to 60. There will be three sets of classes or groups.

**Two invalid classes will be:**

a) Less than or equal to 17.

b) Greater than or equal to 61.

A valid class will be anything between 18 and 60.

We have thus reduced the test cases to only 3 test cases based on the formed classes thereby covering all the possibilities. So, testing with any one value from each set of the class is sufficient to test the above scenario.

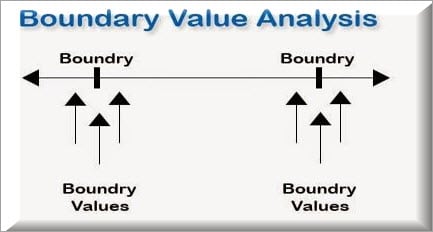
***Recommended Read =>***[***What is Equivalence Partitioning?***](https://www.softwaretestinghelp.com/what-is-boundary-value-analysis-and-equivalence-partitioning/)

#### #2) Boundary Value Analysis

The name itself defines that in this technique, we focus on the values at boundaries as it is found that many applications have a high amount of issues on the boundaries.

Boundary refers to values near the limit where the behavior of the system changes. In boundary value analysis, both valid and invalid inputs are being tested to verify the issues.

**For Example:**

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2018/03/Boundary-Value-Analysis.jpg)

If we want to test a field where values from 1 to 100 should be accepted, then we choose the boundary values: 1-1, 1, 1+1, 100-1, 100, and 100+1. Instead of using all the values from 1 to 100, we just use 0, 1, 2, 99, 100, and 101.

#### #3) Decision Table Testing

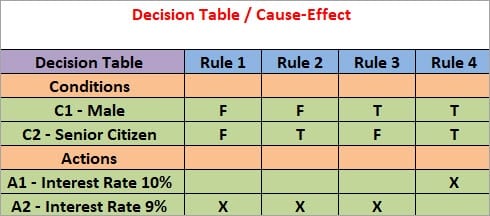
As the name itself suggests, wherever there are logical relationships like:

If  
{  
(Condition = True)  
then action1 ;  
}  
else action2; /\*(condition = False)\*/

Then a tester will identify two outputs (action1 and action2) for two conditions (True and False). So based on the probable scenarios a Decision table is carved to prepare a set of test cases.

**For Example:**

Take an example of XYZ bank that provides an interest rate for the Male senior citizen as 10% and 9% for the rest of the people.

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2018/03/Decision-Table.jpg)

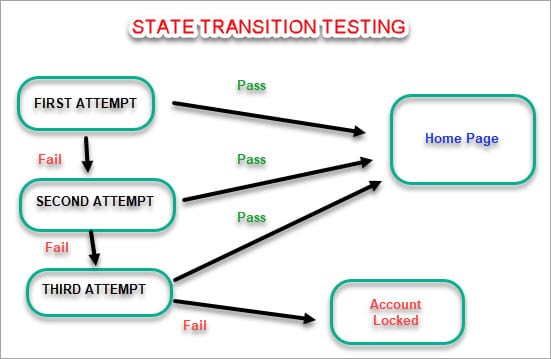
In this example condition, C1 has two values as true and false, C2 also has two values as true and false. The total number of possible combinations would then be four. This way we can derive test cases using a decision table.

#### #4) State Transition Testing

State Transition Testing is a technique that is used to test the different states of the system under test. The state of the system changes depending upon the conditions or events. The events trigger states which become scenarios and a tester needs to test them.

A systematic state transition diagram gives a clear view of the state changes but it is effective for simpler applications. More complex projects may lead to more complex transition diagrams thereby making it less effective.

**For Example:**

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2018/03/State-Transition-Testing.jpg)

#### #5) Error Guessing

This is a classic example of Experience-Based Testing.

In this technique, the tester can use his/her experience about the application behavior and functionalities to guess the error-prone areas. Many defects can be found using error guessing where most of the developers usually make mistakes.

**Few common mistakes that developers usually forget to handle:**

* Divide by zero.
* Handling null values in text fields.
* Accepting the Submit button without any value.
* File upload without attachment.
* File upload with less than or more than the limit size.

**#6) Graph-Based Testing Methods**

Each and every application is a build-up of some objects. All such objects are identified and the graph is prepared. From this object graph, each object relationship is identified and test cases are written accordingly to discover the errors.

**#7) Comparison Testing**

In this method, different independent versions of the same software are used to compare to each other for testing.

### How do I do Step-wise?

In general, when a systematic process is followed to test a project/application then quality is maintained and is useful in the long run for further rounds of testing.

* The foremost step is to understand the requirement specification of an application. Properly documented SRS (Software Requirement Specification) should be in place.
* Using the above mentioned Black Box Testing techniques such as Boundary Value Analysis, Equivalence partitioning etc, sets of valid and invalid inputs are identified with their desired outputs and test cases are designed based on that.
* The designed test cases are executed to check if they Pass or Fail by verifying the actual results with the expected results.
* Failed test cases are raised as Defects/Bugs and addressed to the development team to get it Fixed.
* Further, based on the defects being fixed, the tester retests the defects to verify if they are recurring or not.

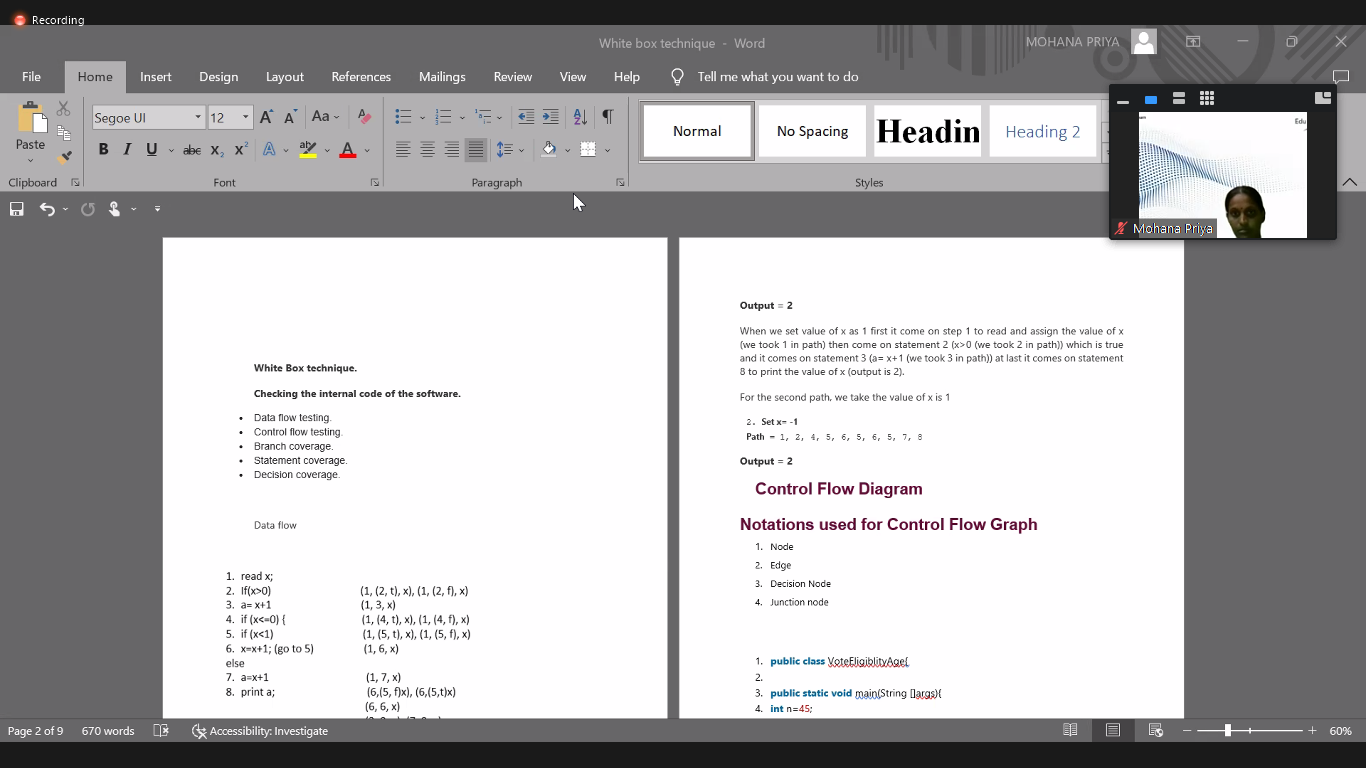
### Advantages and Disadvantages

**Advantages**

* The tester does not need to have a technical background. It is important to test by being in the user’s shoes and think from the user’s point of view.
* Testing can start once the development of the project/application is done. Both the testers and developers work independently without interfering in each other’s space.
* It is more effective for large and complex applications.
* Defects and inconsistencies can be identified in the early stages of testing.

**Disadvantages**

* Without any technical or programming knowledge, there are chances of ignoring possible conditions of the scenario to be tested.
* In a stipulated time there is a possibility of testing less and skipping all possible inputs and their output testing.
* Complete Test Coverage is not possible for large and complex projects



**White box testing**

White box testing techniques analyze the internal structures the used data structures, internal design, code structure and the working of the software rather than just the functionality as in black box testing. It is also called glass box testing or clear box testing or structural testing. White box testing techniques analyze the internal structures the used data structures, internal design, code structure and the working of the software rather than just the functionality as in black box testing. It is also called glass box testing or clear box testing or structural testing.

**Working process of white box testing:**

* **Input:** Requirements, Functional specifications, design documents, source code.
* **Processing:** Performing risk analysis for guiding through the entire process.
* **Proper test planning:** Designing test cases so as to cover entire code. Execute rinse-repeat until error-free software is reached. Also, the results are communicated.
* **Output:** Preparing final report of the entire testing process.

**Testing techniques:**

1. Data Flow Testing.
2. Control flow Testing.
3. Branch Coverage.
4. Statement Coverage.
5. Decision Coverage.

**🡪Data flow testing**

**Data flow testing** is a white-box testing technique that examines the data flow with respect to the variables used in the code. It examines the initialization of variables and checks their values at each instance

**1.** read x, y;

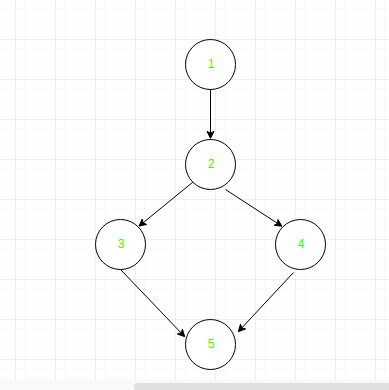
**2.** if(x>y)

**3.** a = x+1

else

**4.** a = y-1

**5.** print a;



|  |  |  |
| --- | --- | --- |
| Variable | Defined at node | Used at node |
| x | 1 | 2, 3 |
| y | 1 | 2, 4 |
| a | 3, 4 | 5 |

### 🡪 Control flow graph

A **control flow graph** is a graphical representation of the flow of control, i.e., the order of statements in which they will be executed.

1.input(x)  
2. if(x>5)  
3.     z = x + 10  
4. else  
5.     z = x - 5  
6. print("Value of Z: ", z)

In the above piece of code, if the value of x entered by the user is greater than 5, then the order of execution of statements would be:

1, 2, 3, 6

If the value entered by the user in line 1 is less than or equal to 5, the order of execution of statements would be:

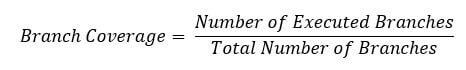
1, 4, 5, 6

## Branch Coverage

**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.

The formula to calculate Branch Coverage:



Demo(int a) {

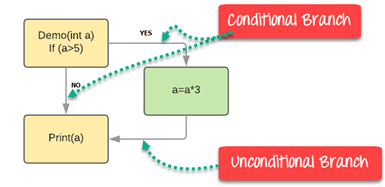
If (a> 5)

a=a\*3

Print (a)

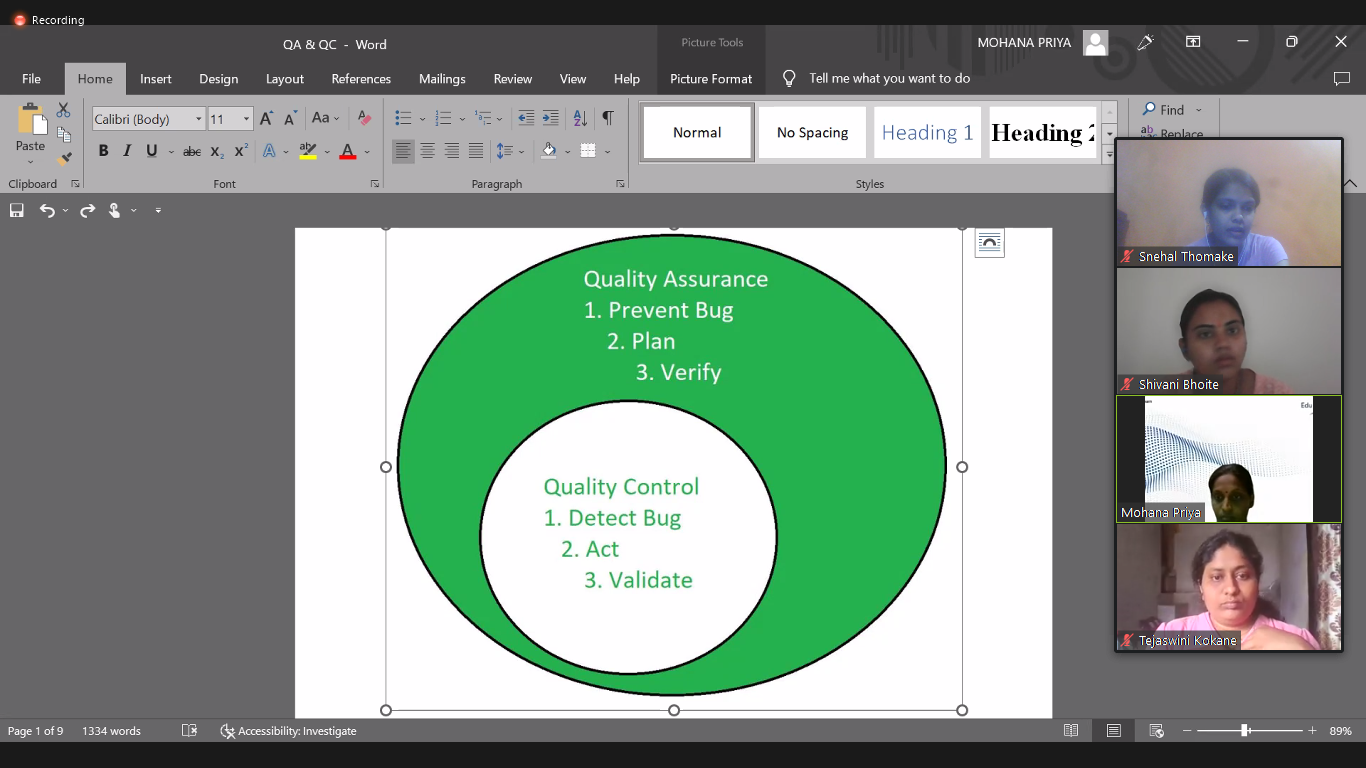
}

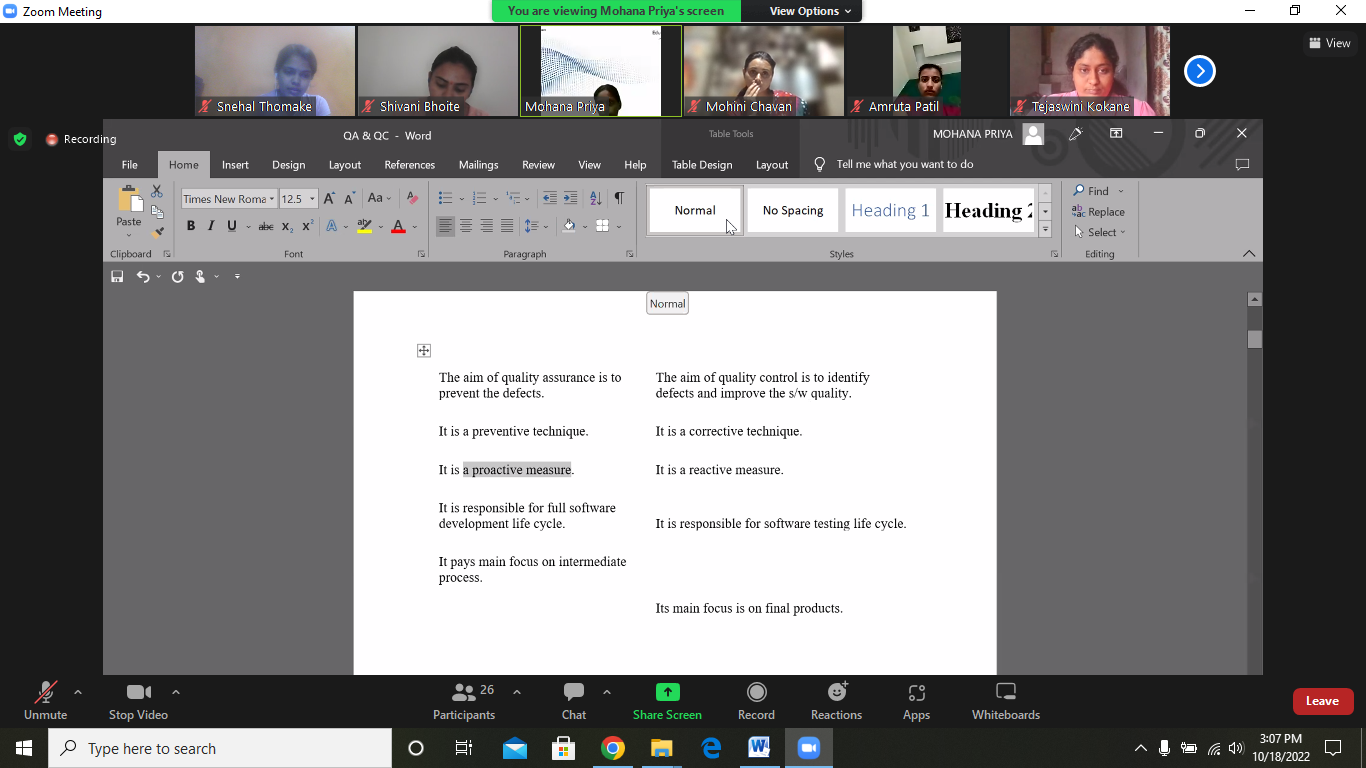
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case** | **Value of A** | **Output** | **Decision Coverage** | **Branch Coverage** |
| 1 | 2 | 2 | 50% | **33%** |
| 2 | 6 | 18 | 50% | **67%** |



### Verification vs Validation Testing: Differences

|  |  |
| --- | --- |
| Verification Testing | Validation Testing |
| It is the static practice of studying and verifying the specific requirements of a particular stage in development. | **It is the dynamic practice of testing the final product after development to check that it meets customer requirements.** |
| It does not require executing code. | **It always requires executing code.** |
| This involves only human verification of required assets. | **This involves both human and machine-based checking and approval of software.** |
| It uses techniques such as document reviews, inspect, product walkthroughs, and desk-checking. | **It involves various types of product testing – unit tests, integration tests, regression tests, cross browser and cross device testing, etc.** |
| It is meant to detect bugs at the beginning of each phase of development. | **It is meant to detect all bugs that were unnoticed at the verification stage.** |
| Its targets are specification documents, the application and software architecture design docs, ER diagrams, database table design, test cases and scenarios, traceability matrix, and the like. | **Its target is the actual product to be used by the customer after public release.** |
| It is undertaken by both developers and testers to ensure that the software adheres to predetermined standards and exactions. | **It is largely undertaken by experienced Quality Assurance engineers who comb through all features of an application to ensure they work as expected.** |
| It comes before validation testing. | **It follows verification testing.** |
| It does not require any devices, platforms, browsers, or operating systems for its execution. | **It is best executed by using real browsers, devices, and operating systems.** |





**There are seven principles in software testing:** 

1. Testing shows the presence of defects
2. Exhaustive testing is not possible
3. Early testing
4. Defect clustering
5. Pesticide paradox
6. Testing is context-dependent
7. Absence of errors fallacy

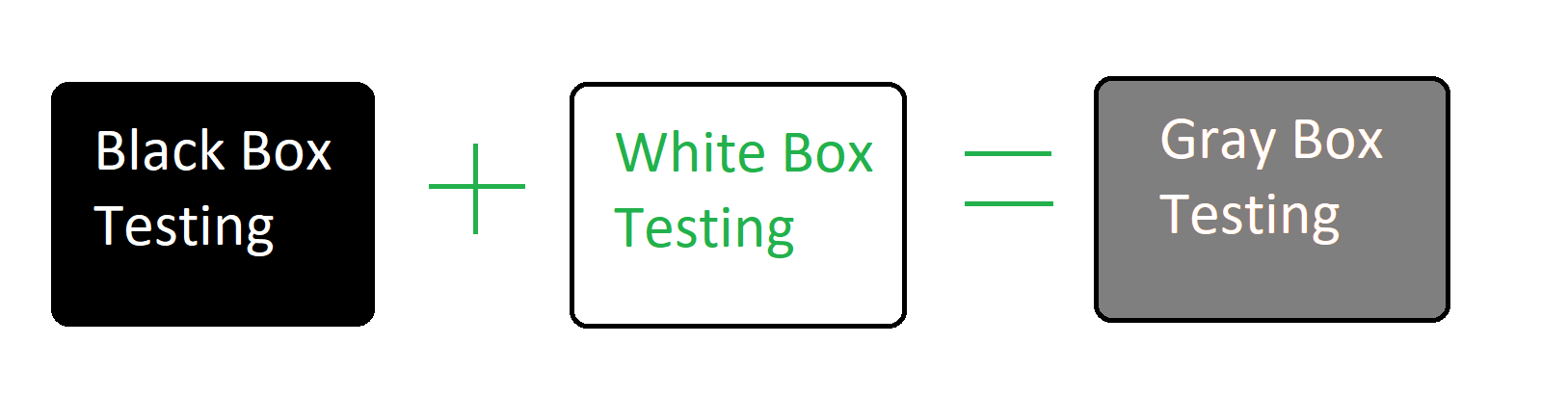
* **Testing shows the presence of defects:** The goal of software testing is to make the software fail. Software testing reduces the presence of defects. Software testing talks about the presence of defects and doesn’t talk about the absence of defects. Software testing can ensure that defects are present but it can not prove that software is defect-free. Even multiple testing can never ensure that software is 100% bug-free. Testing can reduce the number of defects but not remove all defects.
* **Exhaustive testing is not possible:** It is the process of testing the functionality of the software in all possible inputs (valid or invalid) and pre-conditions is known as exhaustive testing. Exhaustive testing is impossible means the software can never test at every test case. It can test only some test cases and assume that the software is correct and it will produce the correct output in every test case. If the software will test every test case then it will take more cost, effort, etc., which is impractical.
* **Early Testing:** To find the defect in the software, early test activity shall be started. The defect detected in the early phases of SDLC will be very less expensive. For better performance of software, software testing will start at the initial phase i.e. testing will perform at the requirement analysis phase.
* **Defect clustering:** In a project, a small number of modules can contain most of the defects. Pareto Principle to software testing state that 80% of software defect comes from 20% of modules.
* **Pesticide paradox:** Repeating the same test cases, again and again, will not find new bugs. So it is necessary to review the test cases and add or update test cases to find new bugs.
* **Testing is context-dependent:** The testing approach depends on the context of the software developed. Different types of software need to perform different types of testing. For example, The testing of the e-commerce site is different from the testing of the Android application.
* **Absence of errors fallacy:** If a built software is 99% bug-free but it does not follow the user requirement then it is unusable. It is not only necessary that software is 99% bug-free but it is also mandatory to fulfill all the customer requirements.

# Date= 18/10/2022

# Gray Box Testing

**Gray Box Testing** is a software testing technique which is a combination of [Black Box Testing](https://www.geeksforgeeks.org/software-engineering-black-box-testing/) technique and [White Box Testing](https://www.geeksforgeeks.org/software-engineering-white-box-testing/) technique. In Black Box Testing technique, tester is unknown to the internal structure of the item being tested and in White Box Testing the internal structure is known to tester. The internal structure is partially known in Gray Box Testing. This includes access to internal data structures and algorithms for purpose of designing the test cases.

Gray Box Testing is named so because the software program is like a semitransparent or grey box inside which tester can partially see.It commonly focuses on context-specific errors related to web systems. It is based on requirement test case generationbecause it has all the conditions presented before the program is tested.



**Objective of Gray Box Testing:**   
The objective of Gray Box Testing is:

1. To provide combined advantages of both black box testing and white box testing.
2. To combine the input of developers as well as testers.
3. To improve overall product quality.
4. To reduce the overhead of long process of functional and non-functional testings.
5. To provide enough free time to developers to fix defects.
6. To test from the user point of view rather than a designer point of view.

**Gray Box Testing Techniques:**

* **Matrix Testing:**   
  In matrix testing technique, business and technical risks which are defined by the developers in software programs are examined. Developers define all the variables that exist in the program. Each of the variables has an inherent technical and business risk and can be used with varied frequencies during its life cycle.
* **Pattern Testing:** To perform the testing, previous defects are analyzed. It determines the cause of the failure by looking into the code. Analysis template includes reasons for the defect. This helps test cases designed as they are proactive in finding other failures before hitting production.
* **Orthogonal Array Testing:**   
  It is mainly a black box testing technique. In orthogonal array testing, test data have n numbers of permutations and combinations. Orthogonal array testing is preferred when maximum coverage is required when there are very few test cases and test data is large. This is very helpful in testing complex applications.
* **Regression Testing:**   
  Regression testing is testing the software after every change in the software to make sure that the changes or the new functionalities are not affecting the existing functioning of the system. Regression testing is also carried out to ensure that fixing any defect has not affected other functionality of the software.

|  |  |  |  |
| --- | --- | --- | --- |
| S. No. | Black Box Testing | Gray Box Testing | White Box Testing |
| 1. | This testing has Low granularity. | This testing has a medium level of granularity. | This testing has high-level granularity. |
| 2. | It is done by end-users and also done by the tester, developers. | It is done by end-users (called user acceptance testing), also done by testers and developers. | It is generally done by testers and developers. |
| 3. | Here, Internals are not required to be known. | Here, Internals relevant to the testing is known. | Here, the Internal code of the application and database is known. |
| 4. | It is likely to be less exhaustive than the other two. | It is kind of in-between. | Most exhaustive among all three. |
| 5. | It is based on requirements, and test cases on the functional specifications, as the internals are not known. | It provides better variety/depth in test cases on account of high-level knowledge of the internals. | It has the ability to exercise code with a relevant variety of data. |
| 6. | If used algorithm testing, is not suited best for that. | If used algorithm testing is also not suited best for that. | If used algorithm testing, it is suited best for that. |
| 7. | It is suited for functional or business testing. | It is suited for functional or business domain testing deeply. | It is used for all. |
| 8. | This testing involves validating the outputs for given inputs, the application being tested as a black-box technique. | Herein, we have a better variety of inputs and the ability to extract test results from the database for comparison with expected results. | It involves structural testing and enables logic coverage, decisions, etc. within the code. |
| 9. | This is also called Opaque-box testing, Closed-box testing, input-output testing, Data-driven testing, Behavioral, Functional testing | This is also called translucent box testing | This is also called Glass-box testing, Clear-box testing, Design-based testing, Logic-based testing, Structural testing, Code-based testing. |
| 10. | **Black-box test design techniques-**   * Decision table testing * Grapht testing * Equivalence partitioning * Error guessing * Comparison testing * State Transition testing | **Gray box test design techniques-**   * Matrix testing * Regression testing * Pattern testing * Orthogonal Array Testing | **White-box test design techniques-**   * Control flow testing * Data flow testing * Branch testing * Statement coverage * Decision coverage |
| 11. | Black Box testing provides resilience and security against viral attacks. | Gray Box testing does not provide resilience and security against viral attacks. | White Box testing does not provide resilience and security against viral attacks. |

Date= 20/10/2022

## What is the Test Case?

A **Test Case** is a set of actions executed to verify a particular feature or functionality of your software application. A [Test Case](https://www.guru99.com/test-case.html) contains test steps, test data, precondition, and postcondition developed for a specific test scenario to verify any requirement. The test case includes specific variables or conditions, using which a testing engineer can compare expected and actual results to determine whether a software product is functioning as per the requirements of the customer.

## What is a Test Scenario?

A **Test Scenario** is defined as any functionality that can be tested. It is a collective set of test cases which helps the testing team to determine the positive and negative characteristics of the project.

[Test Scenario](https://www.guru99.com/test-scenario.html) gives a high-level idea of what we need to test.

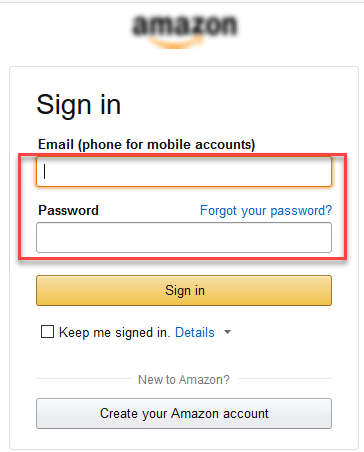
## Example of Test Scenario

For an eCommerce Application, a few test scenarios would be

**Test Scenario 1:**Check the Search Functionality

**Test Scenario 2:**Check the Payments Functionality

**Test Scenario 3:**Check the Login Functionality



## Example of Test Cases

Test cases for the **Test Scenario:** “Check the Login Functionality” would be

1. Check system behavior when valid email id and password is entered.
2. Check system behavior when invalid email id and valid password is entered.
3. Check system behavior when valid email id and invalid password is entered.
4. Check system behavior when invalid email id and invalid password is entered.
5. Check system behavior when email id and password are left blank and Sign in entered.
6. Check Forgot your password is working as expected
7. Check system behavior when valid/invalid phone number and password is entered.
8. Check system behavior when “Keep me signed” is checked

## Test suites

If each test case represents a piece of a scenario, such as the elements that simulate a completing a transaction, use a test suite. For instance, a test suite might contain four test cases, each with a separate test script:

* Test case 1: Login
* Test case 2: Add New Products
* Test case 3: Checkout
* Test case 4: Logout

Test suites can identify gaps in a testing effort where the successful completion of one test case must occur before you begin the next test case. For instance, you cannot add new products to a shopping cart before you successfully log in to the application. When you run a test suite in sequential mode, you can choose to stop the suite execution if a single test case does not pass. Stopping the execution is useful if running a test case in a test suite depends on the success of previous test cases.

Test suites are also useful for the following types of tests:

* Build verification tests: A collection of test cases that perform a basic validation of most the functional areas in the product. The tests are executed after each product build and before the build is promoted for use by a larger audience.
* Smoke tests: A collection of test cases that ensure basic product functionality. Typically, smoke tests are the first level of testing that is performed after changes are made to the system under test.
* End-to-End integration tests: A collection of test cases that cross product boundaries and ensure that the integration points between products are exercised and validated.
* Functional verification tests: A collection of test cases that focus on a specific product function. Executing this type of test with a test suite ensures that several aspects of a specific feature are tested.
* Regression tests: A collection of test cases that are used to make a regression pass over functional product areas.

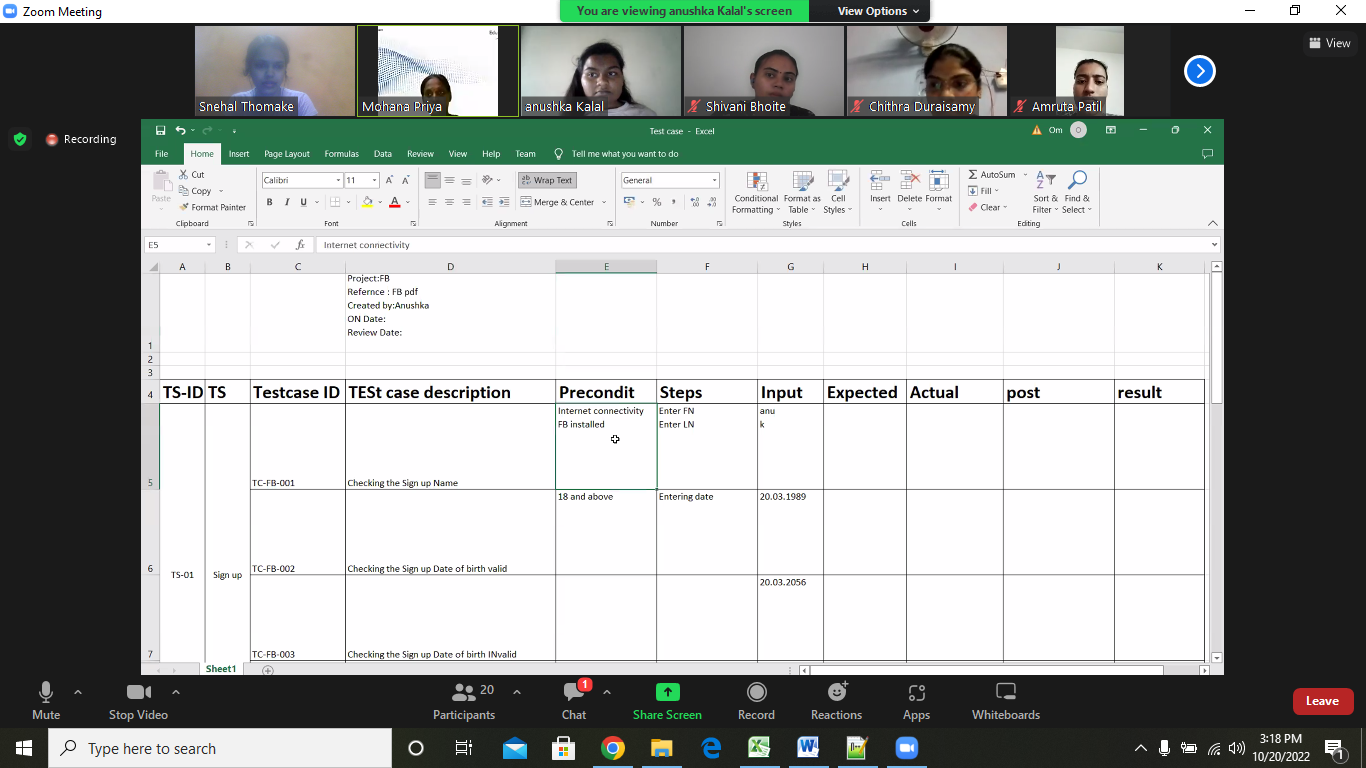
## What is Test Basis?

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

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

## Typical Test Basis:

* Requirement document
* Test Plan
* Codes Repository
* Business Requirement



Snehal Raghunath Thomake

**Date= 21/10/2022**

# Usability Testing

## What is Usability Testing?

Usability testing is the practice of testing how easy a design is to use with a group of representative users. It usually involves observing users as they attempt to complete tasks and can be done for different types of designs. It is often conducted repeatedly, from early development until a product’s release.

**Usability Testing** also known as User Experience (UX) Testing, is a testing method for measuring how easy and user-friendly a software application is. A small set of target end-users, use software application to expose usability defects. Usability testing mainly focuses on user’s ease of using application, flexibility of application to handle controls and ability of application to meet its objectives.

This testing is recommended during the initial design phase of SDLC, which gives more visibility on the expectations of the users.

## Why do Usability Testing

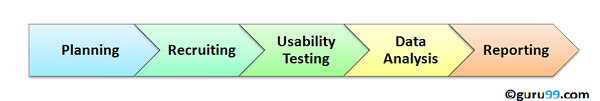
Aesthetics and design are important. How well a product looks usually determines how well it works.

There are many software applications/websites, which miserably fail, once launched, due to following reasons –

* Where do I click next?
* Which page needs to be navigated?
* Which Icon or Jargon represents what?
* Error messages are not consistent or effectively displayed
* Session time not sufficient.

Software Engineering, Usability Testing identifies usability errors in the system early in the development cycle and can save a product from failure.

**Usability Testing Process:**



**1) Planning**:

During this phase the goals of usability test are determined. Having volunteers sit in front of your application and recording their actions is not a goal. You need to determine critical functionalities and objectives of the system. You need to assign tasks to your testers, which exercise these critical functionalities. During this phase, the usability testing method, number & demographics of usability testers, test report formats are also determined

**2) Recruiting**:

During this phase, you recruit the desired number of testers as per your usability test plan. Finding testers who match your demographic (age, sex etc.) and professional ( education, job etc.) profile can take time.

**3) Usability Testing**:

During this phase, usability tests are actually executed.

**4) Data Analysis**:

Data from usability tests is thoroughly analyzed to derive meaningful inferences and give actionable recommendations to improve the overall usability of your product.

**5) Reporting**:

Findings of the usability test is shared with all concerned stakeholders which can include designer, developer, client, and CEO

**Methods of Usability Testing: Techniques**

There are two methods available to do usability testing –

1. Laboratory Usability Testing
2. Remote Usability Testing

**Laboratory Usability Testing:**This testing is conducted in a separate lab room in presence of the observers. The testers are assigned tasks to execute. The role of the observer is to monitor the behavior of the testers and report the outcome of testing. The observer remains silent during the course of testing. In this testing, both observers and testers are present in a same physical location.

**Remote Usability Testing**:

Under this testing observers and testers are remotely located. Testers access the System Under Test, remotely and perform assigned tasks. Tester’s voice , screen activity , testers facial expressions are recorded by an automated software. Observers analyze this data and report findings of the test. Example of such a software – <http://silverbackapp.com/>

**Satisfy User**

The goal of this testing is to satisfy users and it mainly concentrates on the following parameters of a system:

**The effectiveness of the system**

* Is the system is easy to learn?
* Is the system useful and adds value to the target audience?
* Are Content, Color, Icons, Images used are aesthetically pleasing?

**Efficiency**

* Little navigation should be required to reach the desired screen or webpage, and scrollbars should be used infrequently.
* Uniformity in the **format** of screen/pages in your application/website.
* Option to search within your software application or website.

**Accuracy**

* No outdated or incorrect data like contact information/address should be present.
* No broken links should be present.

**User Friendliness**

* Controls used should be self-explanatory and must not require training to operate
* Help should be provided for the users to understand the application/website
* Alignment with the above goals helps in effective usability testing

## Usability Testing Advantages

As with anything in life, usability testing has its merits and de-merits. Let’s look at them

* It helps uncover usability issues before the product is marketed.
* It helps improve end-user satisfaction
* It makes your system highly effective and efficient
* It helps gather true feedback from your target audience who actually use your system during a usability test. You do not need to rely on “opinions” from random people.

## Usability Testing Disadvantages

* Cost is a major consideration in usability testing. It takes lots of resources to set up a Usability Test Lab. Recruiting and management of usability testers can also be expensive

**Security Testing**

**Security Testing** is a type of [Software Testing](https://www.geeksforgeeks.org/software-testing-basics/) that uncovers vulnerabilities of the system and determines that the data and resources of the system are protected from possible intruders. It ensures that the software system and application are free from any threats or risks that can cause a loss. Security testing of any system is focused on finding all possible loopholes and weaknesses of the system which might result in the loss of information or repute of the organization.

**Goal of Security Testing:** The goal of security testing is to:

* To identify the threats in the system.
* To measure the potential vulnerabilities of the system.
* To help in detecting every possible security risks in the system.
* To help developers in fixing the security problems through coding.

**Principle of Security Testing:**

Below are the six basic principles of security testing:

* Confidentiality
* Integrity
* Authentication
* Authorization
* Availability
* Non-repudiation

**Major Focus Areas in Security Testing:**

* Network Security
* System Software Security
* Client-side Application Security
* Server-side Application Security

**Types of Security Testing:**

1. **Vulnerability Scanning:**

Vulnerability scanning is performed with the help of automated software to scan a system to detect the known vulnerability patterns.

1. **Security Scanning:**

Security scanning is the identification of network and system weaknesses. Later on it provides solutions for reducing these defects or risks. Security scanning can be carried out in both manual and automated ways.

1. **Penetration Testing:**

Penetration testing is the simulation of the attack from a malicious hacker. It includes an analysis of a particular system to examine for potential vulnerabilities from a malicious hacker that attempts to hack the system.

1. **Risk Assessment:**

In risk assessment testing security risks observed in the organization are analyzed. Risks are classified into three categories i.e., low, medium and high. This testing endorses controls and measures to minimize the risk.

1. **Security Auditing:**

Security auditing is an internal inspection of applications and operating systems for security defects. An audit can also be carried out via line-by-line checking of code.

1. **Ethical Hacking:**

Ethical hacking is different from malicious hacking. The purpose of ethical hacking is to expose security flaws in the organization’s system.

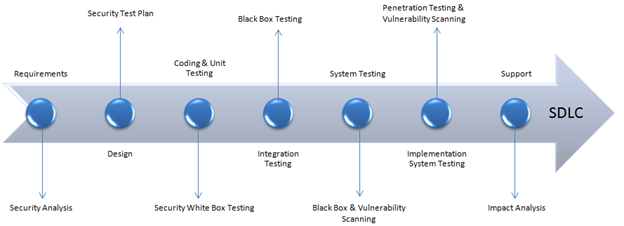
1. **Posture Assessment:**

 It combines security scanning, ethical hacking and risk assessments to provide an overall security posture of an organization.

## How to do Security Testing

It is always agreed, that cost will be more if we postpone security testing after software implementation phase or after deployment. So, it is necessary to involve security testing in the SDLC life cycle in the earlier phases.

Let’s look into the corresponding Security processes to be adopted for every phase in SDLC



|  |  |
| --- | --- |
| SDLC Phases | Security Processes |
| **Requirements** | Security analysis for requirements and check abuse/misuse cases |
| **Design** | Security risks analysis for designing. Development of[Test Plan](https://www.guru99.com/what-everybody-ought-to-know-about-test-planing.html)including security tests |
| **Coding and Unit Testing** | Static and Dynamic Testing and Security [White Box Testing](https://www.guru99.com/white-box-testing.html) |
| **Integration Testing** | [Black Box Testing](https://www.guru99.com/black-box-testing.html) |
| **System Testing** | Black Box Testing and Vulnerability scanning |
| **Implementation** | [Penetration Testing](https://www.guru99.com/learn-penetration-testing.html), Vulnerability Scanning |
| **Support** | Impact analysis of Patches |

**# FUNCTIONAL TESTINGS:**

1.Assertion testing

2.Gorilla testing

3.Sanity testing

4.Monkey testing

5.Smoke testing

6.Exploratory testing

7.Mutation testing

8.Benchmark testing

**1.ASSERTION TESTING:**

**Definition:**

An **assertion** is a **boolean expression**.it is a concept of **functional testing**. It is used to test a **logical expression**. An assertion is true if the logical expression that is being tested is true and there are no bugs in the program. Assertion testing can be used at any particular stage of the program.

EX: 3<4=true, assertion true and logical expression true then their is no bugs.

7>10=false, assertion false and logical expression false then bugs are their.

**Benefits of Assertions:**

The main advantage of having assertions is to identify defects in a program. The usefulness of assertions include:

* It is used to detect subtle errors which might go unnoticed.
* It is used to detect errors sooner after they occur.
* Make a statement about the effects of the code that is guaranteed to be true.

**Limitations:**

* Failing to report a bug that exists.
* Reporting an error when it does not exist.
* Can lead to other side effects
* Can Take time to execute if it contains errors and occupies memory as well.

**2.GORILLA TESTING:**

Gorilla testing is a software testing technique that repeatedly applies inputs on a module to ensure it is functioning correctly and that there are no bugs.

**3.SANITY TESTING:**

Sanity testing is performed on stable builds and it is also known as a variant of regression testing.

Sanity testing was performed when we are receiving software build (with minor code changes) from the development team. It is a checkpoint to assess if testing for the build can proceed or not.

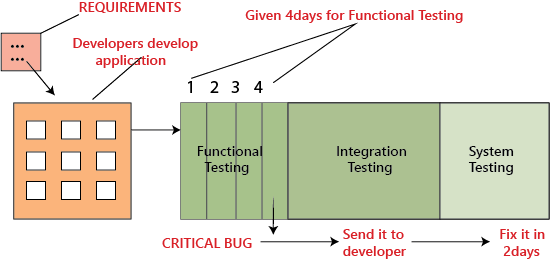
In other words, we can say that sanity testing is performed to make sure that all the defects have been solved and no added issues come into the presence because of these modifications.

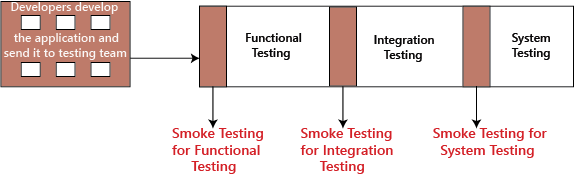
**4.MONKEY TSETING:**

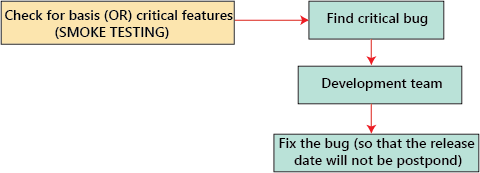
Monkey Testing is a software testing technique in which the tester enters any random inputs into the software application without predefined test cases and checks the behavior of the software application, whether it crashes or not. The purpose of Monkey testing is to find the bugs and errors in the software application using experimental techniques.

**5.SMOKE TESTING:**

Smoke Testing is a software testing process that determines whether the deployed software build is stable or not. Smoke testing is a confirmation for QA team to proceed with further software testing. It consists of a minimal set of tests run on each build to test software functionalities. Smoke testing is also known as “Build Verification Testing” or “Confidence Testing.”







**6.EXPLORATOY TESTING:**

exploratory testing?

If requirement does not exist, then we do one round of exploratory testing.

So, for this first, we will be exploring the application in all possible ways, understanding the flow of the application, preparing a test document and then testing the application, this approach is known as exploratory testing.

**7.MUTATION TESTING:**

Mutation Testing is a type of software testing in which certain statements of the source code are changed/mutated to check if the test cases are able to find errors in source code. The goal of Mutation Testing is ensuring the quality of test cases in terms of robustness that it should fail the mutated source code.

**8. Fuzz Testing** or Fuzzing is a software testing technique of putting invalid or random data called FUZZ into software system to discover coding errors and security loopholes. The purpose of fuzz testing is inserting data using automated or semi-automated techniques and testing the system for various exceptions like system crashing or failure of built-in code, etc.

**# NON-FUNCTIONAL TESTINGS:**

1.Load testing

2.Strees testing

3.Usability testing

4.Performance testing

5.Volume testing

6.Scalability testing

7.Security testing

**1.LOAD TSETING:**

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

**Load Testing Tools:**

1. Apache JMeter

2. Web Load

3. Neo Load

4. Load Ninja

5. HP Performance Tester

6. Load UI Pro

7. Load View

**2.STRESS TESTING:**

Stress testing (sometimes called torture testing) is a form of deliberately intense or thorough testing used to determine the stability of a given system, critical infrastructure or entity. Stress testing involves testing the application under varying load. Extremely large numbers of concurrent users try to log into the application. Database linked to the website shuts down when the website tries to reach it from the front end. Data in added in extremely large quantity in the database. Stress Testing is a type of software testing that verifies stability & reliability of software application. The goal of Stress testing is measuring software on its robustness and error handling capabilities under extremely heavy load conditions and ensuring that software doesn’t crash under crunch situations.

**3.USABILITY TSETING:**

Usability testing refers to evaluating a product or service by testing it with representative users. Typically, during a test, participants will try to complete typical tasks while observers watch, listen and takes notes.

to check the usability or ease of using a software product. Checking the user-friendliness, efficiency, and accuracy of the application is known as **Usability Testing.**

**Parameters:**

Efficiency

Memorability

Accuracy

Learnability

Satisfaction

Errors

**4.PERFORMANCE TESTING:**

Performance testing is in general a testing practice performed to determine how a system performs in terms of responsiveness and stability under a particular workload. It can also serve to investigate, measure, validate or verify other quality attributes of the system, such as scalability, reliability and resource usage.

Types of Performance Testing:

• Load

• Stress

• Spike

• Endurance

• Scalability

• Volume

**5.VOLUME TESTING:**

Volume Testing is a type of software testing which is carried out to test a software application with a certain amount of data.

.

In volume testing a huge volume of data is acted upon the software. It is basically performed to analyze the performance of the system by increasing the volume of data in the database. Volume testing is performed to study the impact on response time and behavior of the system when the volume of data is increased in the database.

Volume Testing is also known as Flood Testing.

**6.SCALABILITY TESTING:**

scalability testing, which comes under the non-functional testing of software testing.

It is used to check an application's performance by increasing or decreasing the load in particular scales known as scalability testing. It is executed at a hardware, software, or database level.

**7.SECURITY TESTING:**

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

types of security Vulnerability Scanning. ...

Security Scanning. ...

Penetration Testing. ...

Security Audit/ Review. ...

Ethical Hacking. ...

Risk Assessment. ...

Posture Assessment. ...

Authentication.

**8. BENCHMARK TESTING:**

A Benchmark in Performance Testing is a metric or a point of reference against which software products or services can be compared to assess the quality measures. In other words, Benchmark means a set standard that helps to determine the quality of a software product or service.

the user experience cannot be quantified in numbers, but the time a user spends on a webpage due to good UI can be quantified.

Benchmark Testing is not a term related to just software testing, but it also deals with Hardware Testing