K.T.S.P.Madal's

Hutatma Rajguru Mahavidyalaya, Rajgurunagar

Tal-Khed, Dist.-Pune 410505.

TY.BSc (Computer Science)

Semester-VI

Subject- Software Testing Tools

According to new CBCS syllabus w.e.f.2019-2020

Prof.S.V.Patole

Department of Computer Science

Hutatma Rajguru Mahavidyalaya,

Rajgurunagar.

Unit 1. Introduction to Test case Design

What is Software Testing?

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.

Basic Terminologies of Software Testing

Acceptance criteria: the predefined and described set of requirements or conditions that must be met in order for the feature to be released to the public.

Ad-hoc testing: Informal testing that does not have any documentation, tickets or planning.

Agile: Not a person who moves quickly, but rather software development that focuses on tasks at hand to be broken down into short phases of work with frequent reappraisals of the work and adaption of the work.

Automated testing: a testing technique that uses an automation testing tool to write test scripts and automate any of the repetitive tasks, comparing the actual outcome with the expected outcome.

Black-box testing: a testing technique that is conducted by a tester not knowing the structure or design of the product whatsoever.

Blocker: this is a bug that is deemed absolutely critical to fix before release of the product. If it is not fixed, it can possibly ruin the entire product or feature launch.

Bugs: are problems, issues, or defects that operate in the program code. These can be minor or major errors, but irregardless they need to be fixed before the release.

Bug reports: this is a formal way to document bugs, but each testing company will have their own version. They will need these reports to be able to give details about the issue to the developers.

Component testing: is a testing technique that focuses separately on small elements of a system.

Configuration testing: another testing technique that ensures that the system or product can operate under various software and hardware conditions, such as a web application being able to properly work in different browsers.

Defects: are issues that do not meet the acceptance criteria. It might not be a bug, but rather a design or content issue that does not match the requirements set forth by the client.

Expected outcome: this is what the system feature or product is supposed to be doing. The observed, or actual outcome is then compared against the expected outcome to show any deviations from the acceptable criteria.

Exploratory testing: means that there is not a test written, rather a tester checks the system based on their knowledge of the system and will execute tests based on that.

Fail/Failure/Failed: the feature or component did not meet the expected outcome.

Feature: changes made to a system or product in order to add new functionalities or modify already existing ones.

Life-cycle testing: is a range of activities that are implemented during the testing process to ensure that the software quality goals are achieved.

Load testing: is a type of testing that measures the performance of the system under a certain load.

Manual testing: is the testing process of manually testing software for bugs and defects where a tester is the 'end-user'.

Mobile-Device testing: is software testing on mobile devices to ensure that works on both the hardware and software.

Negative testing: is the method of testing where invalid information is inputted into the system in order to see if it can handle incorrect or unwanted results.

Observed outcome: is what the tester encounters within the system or product. It is compared to the expected outcome to see if it matches or deviates away from it.

Why Software Testing?

1. Helps in saving money

The testing of software has a wide array of benefits. The **cost-effectiveness** of the project happens to be one of the top reasons why companies go for **software testing Services**.

2. Security

It is another crucial point why **software testing** should not be taken into consideration.

3.Quality of the product

For ensuring that the specific product comes to life, it should work in accordance with the following.

4. Satisfaction of the customer

The primary objective of the owner of the products is offering the best satisfaction of the customers.

The reasons why it is necessary to opt for **software testing** is due to the fact that it offers the prerequisite and perfect user experience.

5. Enhancing the development process

With the aid of **Quality Assurance**, you can find a wide array of scenarios and errors, for the reproduction of the error.

What is Software Quality

Software quality product is defined in term of its fitness of purpose. That is, a quality product does precisely what the users want it to do. For software products, the fitness of use is generally explained in terms of satisfaction of the requirements laid down in the SRS document. Although "fitness of purpose" is a satisfactory interpretation of quality for many devices such as a car, a table fan, a grinding machine, etc.for software products, "fitness of purpose" is not a wholly satisfactory definition of quality.

Example: Consider a functionally correct software product. That is, it performs all tasks as specified in the SRS document. But, has an almost unusable user interface. Even though it may be functionally right, we cannot consider it to be a quality product.

Why Does Software Have Bug

A software defect (a bug) is an error in a program that causes it to work inaccurately and (or) unpredictably. A software defect definition also contains the difference between the actual and expected result.

As bugs in software are created while writing the code, your software may not execute the pre-planned features, work in a manner that differs from your specifications, or perform actions that it shouldn’t. Such cases are called program failures.

Common Software Errors

Functionality Errors

- [ ] Missing functionality
- [ ] Wrong functionality
- [ ] Doesn't do what the user expects
- [ ] Excessive functionality
- [ ] Too Slow or Too Restrictive

Misleading Info Errors

- [ ] Simple factual errors
- [ ] Spelling errors
- [ ] Confusing feature names
- [ ] Similarly named features
- [ ] Information overload

Common Display Bugs

- [ ] Wrong or partial string displayed
- [ ] Messages displayed for too long or not long enough

Inconsistency Bugs

- [ ] Inconsistent capitalization
- [ ] Inconsistent menu position
- [ ] Inconsistent abbreviation. Ex: Jan, July

Menu Bugs

- [ ] No back or forward
- [ ] Too many paths to the same place
- [ ] You can't get there from here situation

Disaster Prevention Bugs

- [ ] No backup facility

- [ ] No undo support
- [ ] No, "Are you sure?"
- [ ] No incremental save

User Tailorability Bugs

- [ ] Can't turn off the noise or sound
- [ ] Can't tailor to popular hardwares
- [ ] Can't change device initialization or startup
- [ ] Can't turn off automatic saves
- [ ] Can't find out what you did last time. Ex: History.

Error Prevention Bugs

- [ ] Coping with invalid data. Ex: Ignoring text in a Phone # Field
- [ ] Implementation of Checksum for file read / write

- [ ] Protection against using incompatible versions
- [ ] Protection against malicious user or inputs. SQL Attack, Injections, etc.

Boundary Bugs

- [ ] Invisible Boundaries
- [ ] Hardware Boundaries
- [ ] Boundaries in Time
- [ ] Numeric Boundaries

Initial & Later States

- [ ] Failure to set a data item to 0
- [ ] Failure to clear a string or flag
- [ ] Failure to reinitialize
- [ ] Incorrect initialization

- [ ] Failure to initialize loop-control variable

Looping Bugs

- [ ] Infinite loop
- [ ] Wrong starting value for the loop control variable
- [ ] Accidental change of the loop control variable
- [ ] Wrong criterion for ending the loop
- [ ] Commands that do or don't belong inside the loop
- [ ] Improper loop nesting

Switch Case Bugs

- [ ] Missing default
- [ ] Missing case in switch statement
- [ ] Overlapping cases

- [ ] Invalid or impossible cases

Missing Bugs

- [ ] Failure to notice a problem
- [ ] Misreading the screen
- [ ] Failure to report a problem
- [ ] Failure to execute a planned test
- [ ] Failure to use the most "promising" test cases
- [ ] Ignoring programmers' suggestions
- [ ] Failure to notice a problem
- [ ] Misreading the screen
- [ ] Failure to report a problem
- [ ] Failure to execute a planned test

- [ ] Failure to use the most "promising" test cases
- [ ] Ignoring programmers' suggestions

Poor Reporting Bugs

- [ ] Illegible reports
- [ ] Failure to make it clear how to reproduce the problem
- [ ] Failure to say that you can't reproduce a problem
- [ ] Failure to check your report
- [ ] Failure to report timing dependencies
- [ ] Failure to simplify conditions
- [ ] Concentration on trivial issues
- [ ] Abusive or aggressive language

Finding Wrong Bugs

- [ ] Errors in testing programs
- [ ] Corrupted data file
- [ ] Misinterpreted specifications or documentation

How to Identify Errors, Bugs in the given Application?

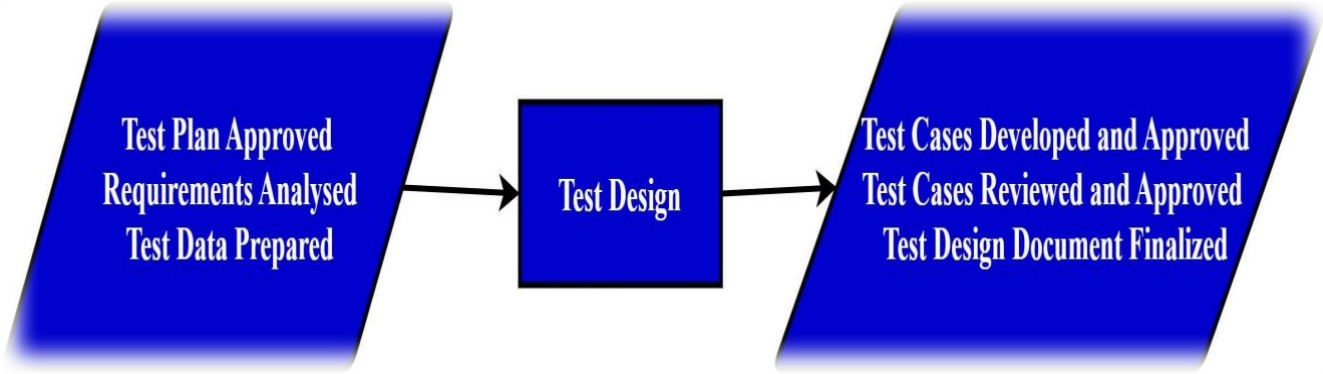
- The recent iPhone bug where users could not type the letter “I”.
- Some are costly bugs and can cost a fortune to fix one such bug: the Y2K bug.
- One software bug literally led to the death of people due to the patriot missile bug; 28 people died in 1991.
- Any buggy code reflects poorly on them and their team and will eventually affect the company’s bottom line.
- Also, buggy code is inconvenient to work with and reduces productivity. The more quality code you can write, the more effective you will be. Finally, bugs are expensive.
- Various software bugs are estimated to have cost the global economy $1.7 trillion in 2017.

Hence, **finding and solving even the small bugs is crucial** even in each and every software. Bugs in software can literally shut down your business, and I am not even kidding.

In the end, if a user is not getting a great product, he will shift, and there are always alternatives. So let us understand in detail how to find bugs.

Design Entry and Exit Criteria for Test Case

This phase also has several entry and exit criteria:



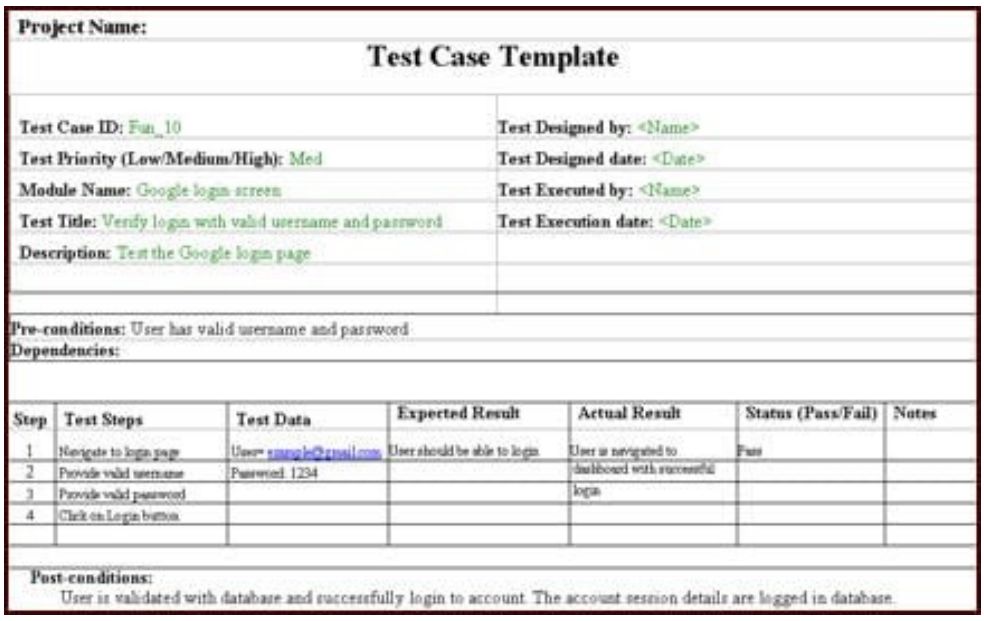
The test plan, approved in the test planning phase, serves as the foundation for test design. It provides us with essential information about testing objectives, scope, and strategies that guide the creation of test cases.

The testing team must thoroughly analyze the project requirements and ensure the test cases cover all specified functionalities. Test data required for executing the test cases should be prepared and available in the testing environment. This data should be relevant to the test scenarios.

The exit criteria of the test design phase include the development, review, and approval of the test cases. Each case should undergo a thorough review process to identify and address any defects or ambiguities before approval. The test design phase is complete when we finalize the test design document containing test cases, test scenarios, and test data and get it ready for execution.

Test Case – Sample Structure

There are certain standard fields that need to be considered while preparing a Test case template.



Project Name:

Test Case Template

Test Case ID: Fun_10	**Test Designed by:** <Name>
Test Priority (Low/Medium/High): Med	**Test Designed date:** <Date>
Module Name: Google login screen	**Test Executed by:** <Name>
Test Title: Verify login with valid username and password	**Test Execution date:** <Date>
Description: Test the Google login page	

Pre-conditions: User has valid username and password
Dependencies:

Step	Test Steps	Test Data	Expected Result	Actual Result	Status (Pass/Fail)	Notes
1	Navigate to login page	User= sample@gmail.com	User should be able to login	User is navigated to	Pass	
2	Provide valid username	Password: 1234		dashboard with successful		
3	Provide valid password			login		
4	Click on Login button					

Post-conditions:
User is validated with database and successfully login to account. The account session details are logged in database

Several standard fields for a sample Test Case template are listed below.

Test case ID: Unique ID is required for each test case. Follow some conventions to indicate the types of the test. **For Example,** 'TC_UI_1' indicating 'user interface test case #1'.

Test priority (Low/Medium/High): This is very useful during test execution. Test priorities for business rules and functional test cases can be medium or higher, whereas minor user

interface cases can be of a low priority. Testing priorities should always be set by the reviewer.

Module Name: Mention the name of the main module or the sub-module.

Test Designed By Name of the Tester.

Test Designed Date: Date when it was written.

Test Executed By Name of the Tester who executed this test. To be filled only after test execution.

Test Execution Date: Date when the test was executed.

Various Types Of Testing Methods

Test Title/Name: Test case title. **For example,** verify the login page with a valid username and password.

Test Summary/Description: Describe the test objective in brief.

Pre-conditions: Any prerequisite that must be fulfilled before the execution of this test case. List all the pre-conditions in order to execute this test case successfully.

Dependencies: Mention any dependencies on other test cases or test requirements.

Test Steps: List all the test execution steps in detail. Write test steps in the order in which they should be executed. Make sure to provide as many details as you can.

Pro Tip: In order to manage a test case efficiently with a lesser number of fields, use this field to describe the test conditions, test data and user roles for running the test.

Test Data: Use of test data as an input for this test case. You can provide different data sets with exact values to be used as an input.

Expected Result: What should be the system output after test execution? Describe the expected result in detail including the message/error that should be displayed on the screen.

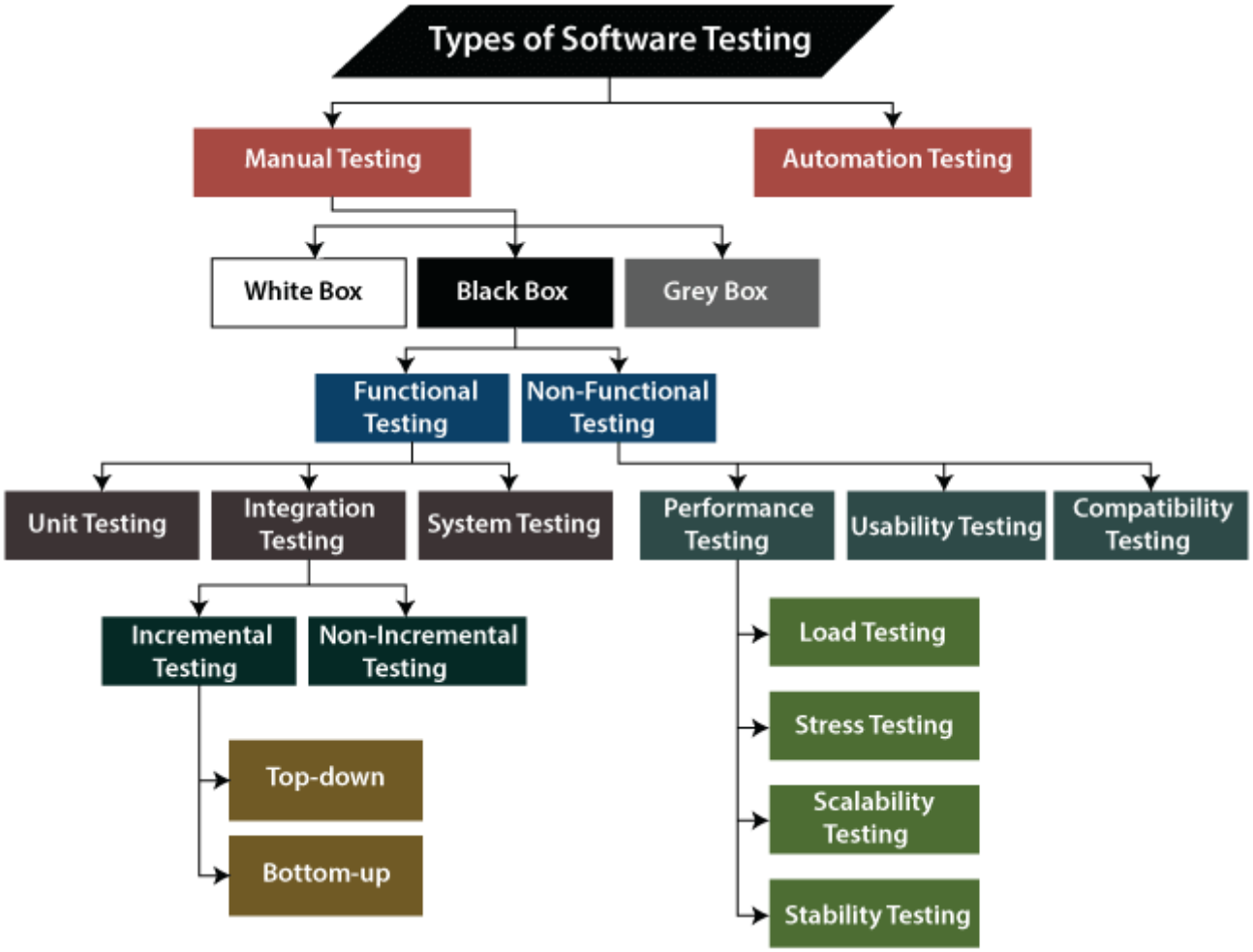
Post-condition: What should be the state of the system after executing this test case?

Actual result: The actual test result should be filled after test execution. Describe the system behavior after test execution.

Status (Pass/Fail): If the actual result is not as per the expected result, then mark this test as **failed**. Otherwise, update it as **passed**.

Various Types of Testing Methods

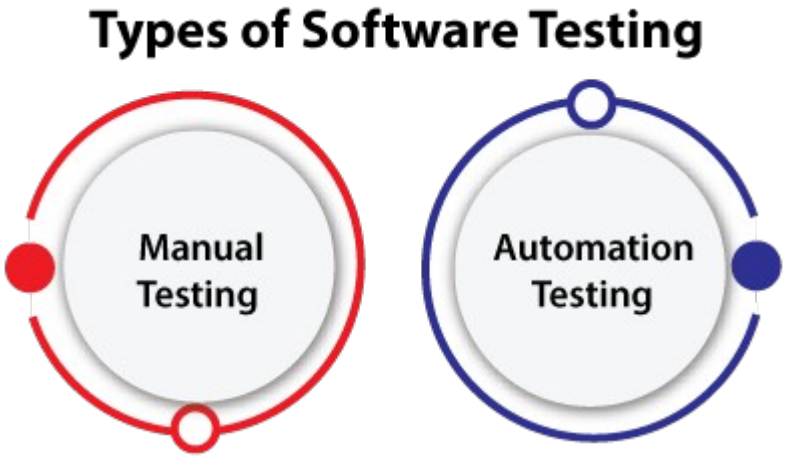
If we want to ensure that our software is bug-free or stable, we must perform the various types of software testing because testing is the only method that makes our application bug-free.



The different types of Software Testing

The categorization of software testing is a part of diverse testing activities, such as **test strategy, test deliverables, a defined test objective, etc**. And software testing is the execution of the software to find defects.

The purpose of having a testing type is to confirm the **AUT** (Application Under Test). To start testing, we should have a **requirement, application-ready, necessary resources available**. To maintain accountability, we should assign a respective module to different test engineers. The software testing mainly divided into two parts, which are as follows:

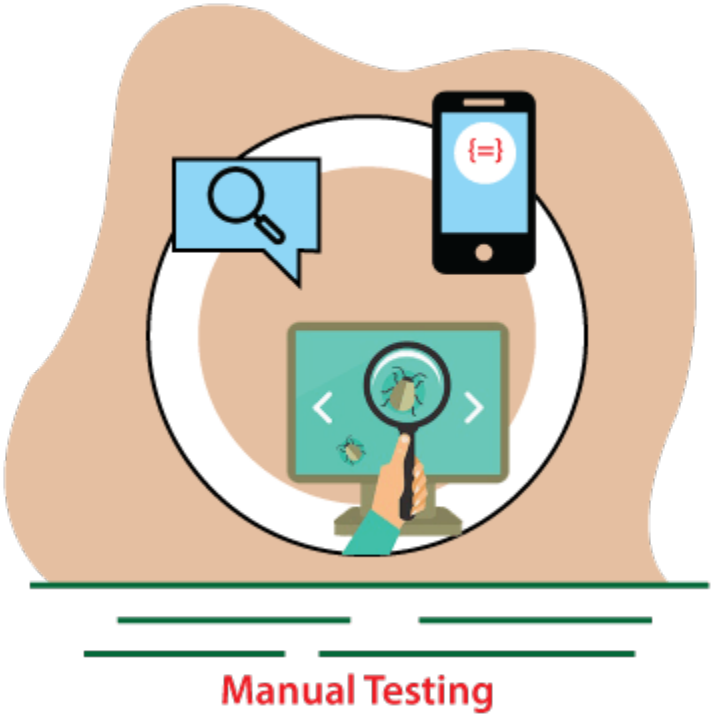


- **Manual Testing**
- **Automation Testing**

What is Manual Testing?

Testing any software or an application according to the client's needs without using any automation tool is known as **manual testing**.

In other words, we can say that it is a procedure of **verification and validation**. Manual testing is used to verify the behavior of an application or software in contradiction of requirements specification.



We do not require any precise knowledge of any testing tool to execute the manual test cases. We can easily prepare the test document while performing manual testing on any application.

To get in-detail information about manual testing, click on the following link: https://www.javatpoint.com/manual-testing.

Classification of Manual Testing

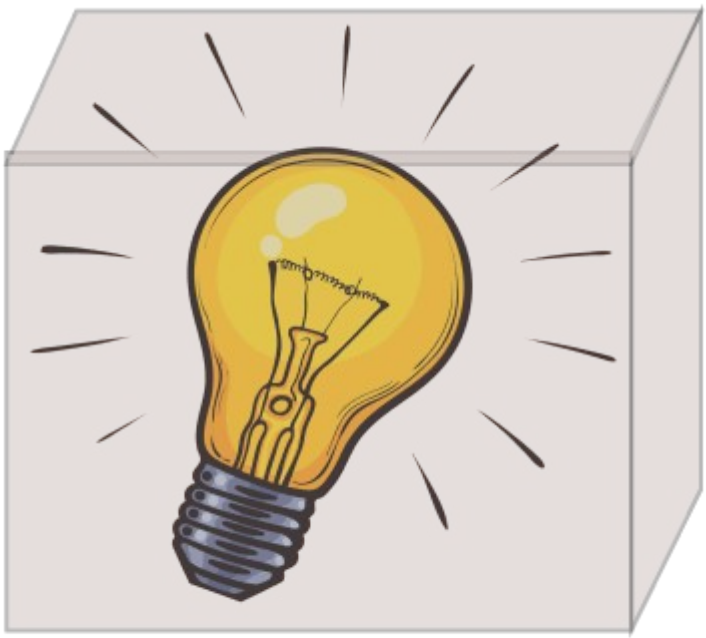
In software testing, manual testing can be further classified into **three different types of testing**, which are as follows:

- **White Box Testing**
- **Black Box Testing**
- **Grey Box Testing**

For our better understanding let's see them one by one:

White Box Testing

In white-box testing, the developer will inspect every line of code before handing it over to the testing team or the concerned test engineers.

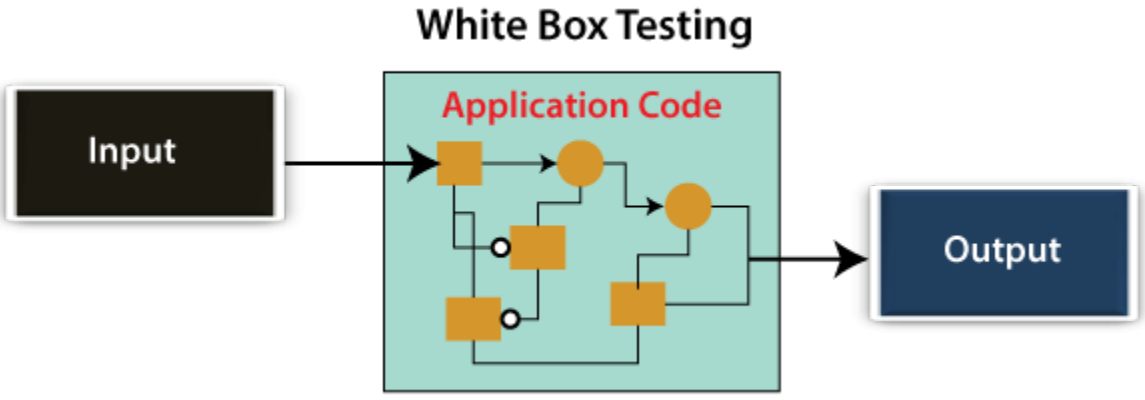


White Box Testing

Subsequently, the code is noticeable for developers throughout testing; that's why this process is known as **WBT (White Box Testing)**.

In other words, we can say that the **developer** will execute the complete white-box testing for the particular software and send the specific application to the testing team.

The purpose of implementing the white box testing is to emphasize the flow of inputs and outputs over the software and enhance the security of an application.



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

To get the in-depth knowledge about white box testing refers to the below link: https://www.javatpoint.com/white-box-testing.

Black Box Testing

Another type of manual testing is **black-box testing**. In this testing, the test engineer will analyze the software against requirements, identify the defects or bug, and sends it back to the development team.

Then, the developers will fix those defects, do one round of White box testing, and send it to the testing team.

Here, fixing the bugs means the defect is resolved, and the particular feature is working according to the given requirement.

The main objective of implementing the black box testing is to specify the business needs or the customer's requirements.

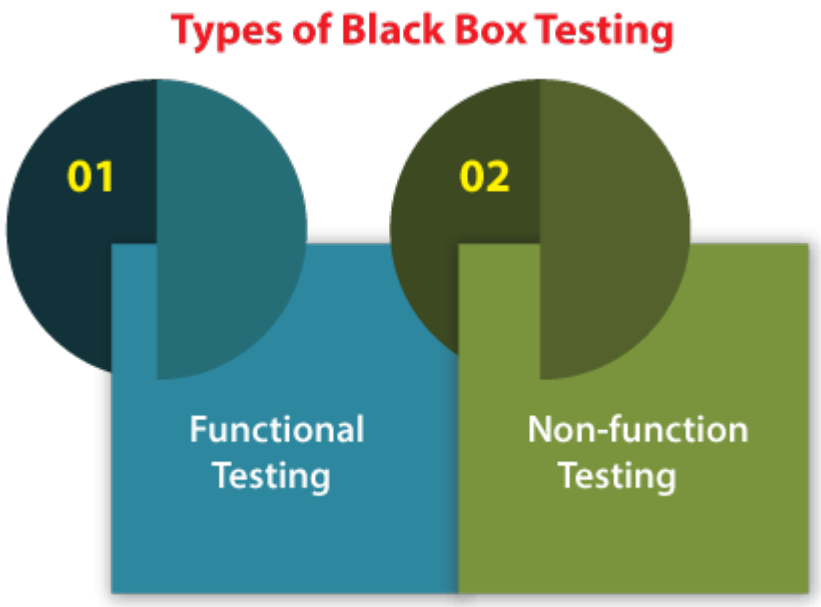
In other words, we can say that black box testing is a process of checking the functionality of an application as per the customer requirement. The source code is not visible in this testing; that's why it is known as **black-box testing**.

For more information about Black box testing, refers to the below link: https://www.javatpoint.com/black-box-testing.

Types of Black Box Testing

Black box testing further categorizes into two parts, which are as discussed below:

- **Functional Testing**
- **Non-function Testing**



Functional Testing

The test engineer will check all the components systematically against requirement specifications is known as **functional testing**. Functional testing is also known as **Component testing**.

In functional testing, all the components are tested by giving the value, defining the output, and validating the actual output with the expected value.

Functional testing is a part of black-box testing as its emphases on application requirement rather than actual code. The test engineer has to test only the program instead of the system.

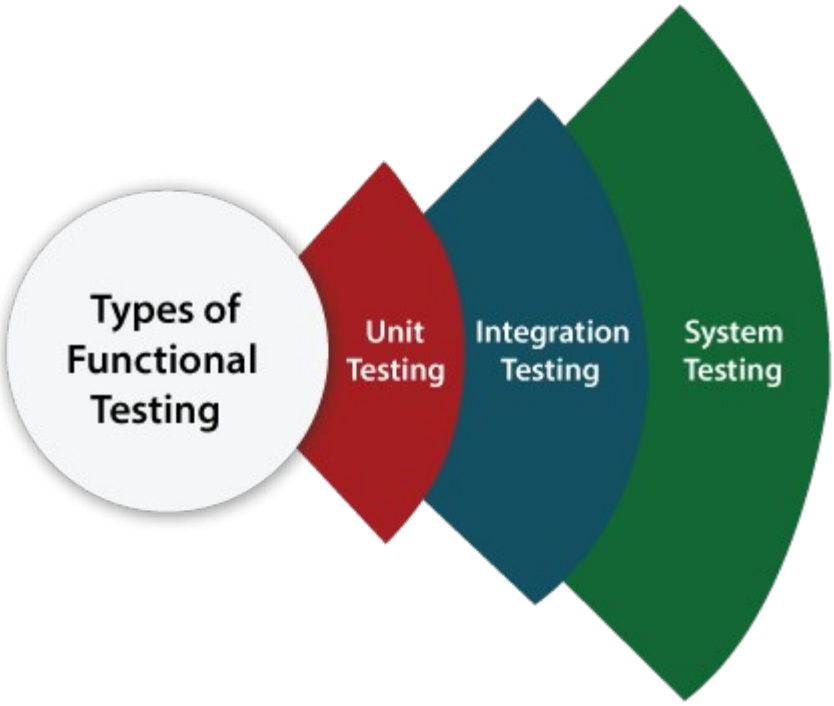
To get the detailed information about functional testing refers to the below link: https://www.javatpoint.com/functional-testing.

Types of Functional Testing

Just like another type of testing is divided into several parts, functional testing is also classified into various categories.

The diverse **types of Functional Testing** contain the following:

- **Unit Testing**
- **Integration Testing**
- **System Testing**



Now, Let's understand them one by one:

1. Unit Testing

Unit testing is the first level of functional testing in order to test any software. In this, the test engineer will test the module of an application independently or test all the module functionality is called **unit testing**.

The primary objective of executing the unit testing is to confirm the unit components with their performance. Here, a unit is defined as a single testable function of a software or an application. And it is verified throughout the specified application development phase.

Click on the below link to get the complete information about unit testing: https://www.javatpoint.com/unit-testing.

2. Integration Testing

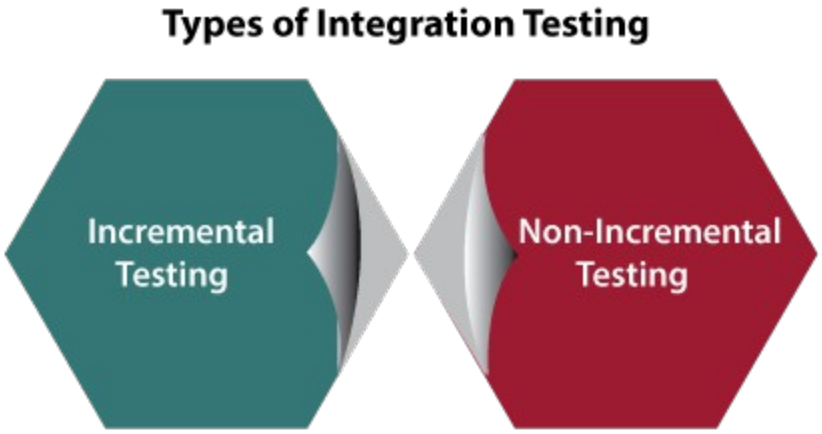
Once we are successfully implementing the unit testing, we will go integration testing. It is the second level of functional testing, where we test the data flow between dependent modules or interface between two features is called **integration testing**.

The purpose of executing the integration testing is to test the statement's accuracy between each module.

Types of Integration Testing

Integration testing is also further divided into the following parts:

- **Incremental Testing**
- **Non-Incremental Testing**



Incremental Integration Testing

Whenever there is a clear relationship between modules, we go for incremental integration testing. Suppose, we take two modules and analysis the data flow between them if they are working fine or not.