White Box Testing

**********

*It tests internal coding and infrastructure of a software focus on checking of predefined inputs against expected and desired outputs.

*White Box Testing which also known as Glass Box testing, Structural Testing, Clear Box Testing, Open Box Testing and Transparent Box Testing.

*The term 'white box' is used because of the internal perspective of the system.

* 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 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

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.

Techniques that are Used in White Box Testing:

1.Statement Coverage Technique

2.Branch coverage Technique

3.Data flow Technique or Path Technique

4.Control flow coverage Technique

5.Decision coverage Technique

1.Statement Coverage Technique:

*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.

*Statement coverage derives scenario of test cases under the white box testing process which is based upon the structure of the code.

Source Code Structure:

Take input of two values like a=0 and b=1.

Find the sum of these two values.

If the sum is greater than 0, then print "This is the positive result."

If the sum is less than 0, then print "This is the negative result."

Program:

```
input (int a, int b)
{
Function to print sum of these integer values (sum = a+b)
If (sum>0)
{
Print (This is positive result)
} else
{
Print (This is negative result)
}
}
```

Scenario1:

If a = 5, b = 4

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

In scenario 1, we can see the value of sum will be 9 that is greater than 0 and as per the condition result will be "This is a positive result." The statements highlighted in yellow color are executed statements of this scenario.

To calculate statement coverage of the first scenario, take the total number of statements that is 7 and the number of used statements that is 5.

Total number of statements = 7

Number of executed statements = 5

Statement coverage = 5/7*100

= 500/7

= 71%

Scenario2:

If A = -2, B = -7

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

In scenario 2, we can see the value of sum will be -9 that is less than 0 and as per the condition, result will be "This is a negative result." The statements highlighted in yellow color are executed statements of this scenario.

To calculate statement coverage of the first scenario, take the total number of statements that is 7 and the number of used statements that is 6.

Total number of statements= 7

Number of executed statements = 6

Statement coverage = 6/7*100 <br>

= 600/7

= 85%

But, we can see all the statements are covered in both scenario and we can consider that the overall statement coverage is 100%.

So, the statement coverage technique covers dead code, unused code, and branches.

2.Branch Coverage Technique:

*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.

*Branch coverage technique is a whitebox testing technique that ensures that every branch of each decision point must be executed.

How to calculate Branch coverage:

Example:

Read X

Read Y

IF X+Y > 100 THEN

Print "Large"

ENDIF

If X + Y<100 THEN

Print "Small"

ENDIF

This is the basic code structure where we took two variables X and Y and two conditions. If the first condition is true, then print "Large" and if it is false, then go to the next condition. If the second condition is true, then print "Small."

Control flow graph of code structure:

In the above diagram, control flow graph of code is depicted. In the first case traversing through "Yes "decision, the path is A1-B2-C4-D6-E8, and the number of covered edges is 1, 2, 4, 5, 6 and 8 but edges 3 and 7 are not covered in this path. To cover these edges, we have to traverse through "No" decision. In the case of "No" decision the path is A1-B3-5-D7, and the number of covered edges is 3 and 7. So by traveling through these two paths, all branches have covered.

Path 1 - A1-B2-C4-D6-E8

Path 2 - A1-B3-5-D7

Branch Coverage (BC) = Number of paths

=2

Case

Covered Branches

Path

Branch coverage

Yes

1, 2, 4, 5, 6, 8

A1-B2-C4-D6-E8

2

No

3,7

A1-B3-5-D7

3.Data flow Technique or Path Technique:

*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.

Example:

1. x= 1

Path - 1, 2, 3, 8

Output = 2

When we set value of x as 1 first it come on step 1 to read and assign the value of x (we took 1 in path) then come on statement 2 (x>0 (we took 2 in path)) which is true and it comes on statement 3 (a= x+1 (we took 3 in path)) at last it comes on statement 8 to print the value of x (output is 2).

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

Output = 2

When we set the value of x as ?1 then first, it comes on step 1 to read and assign the value of x (we took 1 in the path) then come on step number 2 which is false because x is not greater than 0 (x>0 and their x=-1). Due to false condition, it will not come on statement 3 and directly jump on statement 4 (we took 4 in path) and 4 is true (x<=0 and their x is less than 0) then come on statement 5 (x<1 (we took 5 in path)) which is also true so it will come on statement 6 (x=x+1 (we took 6 in path)) and here x is incremented by 1.

4.Control Flow Testing:

Control flow testing is a testing technique that comes under white box testing. The aim of this technique is to determine the execution order of statements or instructions of the program through a control structure.

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

Notations used for Control Flow Graph

Node

Edge

Decision Node

Junction node

Example:

```
public class VoteEligiblityAge{


public static void main(String []args){

int n=45;

if(n>=18)

{

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

}    else

{

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

}

}

}
```

5.Decision Coverage Testing:

Decision coverage technique comes under white box testing which gives decision coverage to Boolean values. This technique reports true and false outcomes of Boolean expressions. Whenever there is a possibility of two or more outcomes from the statements like do while statement, if statement and case statement (Control flow statements), it is considered as decision point because there are two outcomes either true or false.

Let's understand it by an example.

Consider the code to apply on decision coverage technique:

```
Test (int a)

{

If(a>4)

a=a*3

Print (a)

}
```

Scenario 1:

Value of a is 7 (a=7)

```
Test (int a=7)

{ if (a>4)

a=a*3

print (a)

}
```

The code highlighted in yellow is executed code. The outcome of this code is "True" if condition (a>4) is checked.

Control flow graph when the value of a is 7.

**Test Basics, Test Case, Test Suite, Test Scenario**

Test Basics:

What is Testing basics?

*Testing is the process of evaluating a system or its component(s) with the intent to find whether it satisfies the specified requirements or not.

*Testing is executing a system in order to identify any gaps, errors, or missing requirements in contrary to the actual requirements.

* Software testing can be stated as the process of verifying and validating.

* 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: 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.

Clue's for Verification and Validation:

Verification : "Are we building the product right?"

Validation : "Are we building the right product?"

What are different 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.

*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.

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

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.

The different types of Software Testing Techniques:

Software testing techniques can be majorly classified into two categories:

1. Black Box Testing

2. White-Box Testing

What are different levels of software testing?

Software level testing can be majorly classified into 4 levels:

1. Unit Testing

2. Integration Testing

3. System Testing

4. Acceptance Testing

Test Case:

*A Test Case refers to the actions required to verify a specific feature or functionality in software testing.

*The test case details the steps, data, prerequisites, and postconditions necessary to verify a feature.

Test Suite:

*Test suite is a container that has a set of tests which helps testers in executing and reporting the test execution status.

*It can take any of the three states namely Active, Inprogress and completed.

*A Test case can be added to multiple test suites and test plans. After creating a test plan, test suites are created which in turn can have any number of tests.

*Test suites are created based on the cycle or based on the scope. It can contain any type of tests, Viz - Functional or Non-Functional.

An Example of a Test Suite:

For instance, a test suite for product purchase has multiple test cases, like: Test Case 1: Login. Test Case 2: Adding Products.

Test Scenario:

A Test Scenario, sometimes called a scenario test, is the documentation of a use case. In other words, it describes an action the user may undertake with a website or app. It may also represent a situation the user may find themselves in while using that software or product.

An Example of a Test Scenario:

A Test Scenario is a description of an objective a user might face when using the program.

An example might be “Test that the user can successfully log out by closing the program.” Typically, a test scenario will require testing in a few different ways to ensure the scenario has been satisfactorily covered.

Test Case Plan

**********

https://www.javatpoint.com/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.

*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%).

Types of Test Plan:

There are three types of the test plan:

1.Master Test Plan

2.Phase Test Plan

3.Testing Type Specific Test Plans

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.

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.

3.Specific Test Plans:

*Specific Test Plan designed for major types of testing like security testing, load testing, performance testing, etc.

*In other words, a specific test plan designed for non-functional testing.

How to write a Test Plan:

*Making a test plan is the most crucial task of the test management process.

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

1.First, analyze product structure and architecture.

2.Now design the test strategy.

3.Define all the test objectives.

4.Define the testing area.

5.Define all the useable resources.

6.Schedule all activities in an appropriate manner.

7.Determine all the Test Deliverables.

8.Test plan components or attributes

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

Test plan components or attributes:

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

Objectives:

*It consists of information about modules, features, test data etc.,

*Which indicate the aim of the application means the application behavior, goal, etc.

Scope:

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

In scope

Out scope

In scope:

These are the modules that need to be tested rigorously (in-detail).

Out scope:

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

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

## Software Testing Life Cycle (STLC)

**********

What is STLC in testing?

*STLC stands for Software Testing Life Cycle.

*Software Testing Life Cycle (STLC) is a process used to test software and ensure that quality standards are met.

*STLC is a sequence of different activities performed by the testing team to ensure the quality of the software or the product.

*STLC is an integral part of Software Development Life Cycle (SDLC).

Characteristics of STLC:

*STLC is a fundamental part of the Software Development Life Cycle (SDLC) but STLC consists of only the testing phases.

*STLC starts as soon as requirements are defined or software requirement document is shared by stakeholders.

*STLC yields a step-by-step process to ensure quality software.

Phases of STLC:

1. Requirement Analysis

2. Test Planning

3. Test Case Development

4. Test Environment Setup

5. Test Execution

6. Test Closure

1. Requirement Analysis:

*Requirement Analysis is the first step of the Software Testing Life Cycle (STLC).

*In this phase quality assurance team understands the requirements like what is to be tested.

*If anything is missing or not understandable then the quality assurance team meets with the stakeholders to better understand the detailed knowledge of requirements.

The activities that take place during the Requirement Analysis stage include:

1.Reviewing the Software Requirements Document (SRD) and other related documents.

2.Interviewing stakeholders to gather additional information.

3.Identifying any ambiguities or inconsistencies in the requirements.

4.Identifying any missing or incomplete requirements.

5.Identifying any potential risks or issues that may impact the testing process.

6.Creating a Requirement Traceability Matrix (RTM) to map requirements to test cases.

*At the end of this stage, the testing team should have a clear understanding of the software requirements and should have identified any potential issues that may impact the testing process.

*This will help to ensure that the testing process is focused on the most important areas of the software and that the testing team is able to deliver high-quality results.

2. Test Planning:

*Test Planning is the most efficient phase of the software testing life cycle where all testing plans are defined.

*In this phase manager of the testing, team calculates the estimated effort and cost for the testing work. This phase gets started once the requirement-gathering phase is completed.

The activities that take place during the Test Planning stage include:

1.Identifying the testing objectives and scope.

2.Developing a test strategy: selecting the testing methods and techniques that will be used.

3.Identifying the testing environment and resources needed.

4.Identifying the test cases that will be executed and the test data that will be used.

5.Estimating the time and cost required for testing.

6.Identifying the test deliverables and milestones.

7.Assigning roles and responsibilities to the testing team.

8.Reviewing and approving the test plan.

At the end of this stage, the testing team should have a detailed plan for the testing activities that will be performed, and a clear understanding of the testing objectives, scope, and deliverables.

This will help to ensure that the testing process is well-organized and that the testing team is able to deliver high-quality results.

3. Test Case Development:

*The test case development phase gets started once the test planning phase is completed.

*In this phase testing team notes down the detailed test cases.

*The testing team also prepares the required test data for the testing. *When the test cases are prepared then they are reviewed by the quality assurance team.

The activities that take place during the Test Case Development stage include:

1.Identifying the test cases that will be developed.

2.Writing test cases that are clear, concise, and easy to understand.

3.Creating test data and test scenarios that will be used in the test cases.

4.Identifying the expected results for each test case.

5.Reviewing and validating the test cases.

6.Updating the requirement traceability matrix (RTM) to map requirements to test cases.

At the end of this stage, the testing team should have a set of comprehensive and accurate test cases that provide adequate coverage of the software or application.

This will help to ensure that the testing process is thorough and that any potential issues are identified and addressed before the software is released.

4. Test Environment Setup:

*Test environment setup is a vital part of the STLC. Basically, the test environment decides the conditions on which software is tested.

*This is independent activity and can be started along with test case development.

*In this process, the testing team is not involved.

*Either the developer or the customer creates the testing environment.

5. Test Execution:

*After the test case development and test environment setup test execution phase gets started.

*In this phase testing team starts executing test cases based on prepared test cases in the earlier step.

The activities that take place during the test execution stage of the Software Testing Life Cycle (STLC) include:

*Test execution: The test cases and scripts created in the test design stage are run against the software application to identify any defects or issues.

*Defect logging: Any defects or issues that are found during test execution are logged in a defect tracking system, along with details such as the severity, priority, and description of the issue.

*Test data preparation: Test data is prepared and loaded into the system for test execution.

*Test environment setup: The necessary hardware, software, and network configurations are set up for test execution.

*Test execution: The test cases and scripts are run, and the results are collected and analyzed.

*Test result analysis: The results of the test execution are analyzed to determine the software's performance and identify any defects or issues.

*Defect retesting: Any defects that are identified during test execution are retested to ensure that they have been fixed correctly.

*Test Reporting: Test results are documented and reported to the relevant stakeholders.

It is important to note that test execution is an iterative process and may need to be repeated multiple times until all identified defects are fixed and the software is deemed fit for release.

6. Test Closure:

*Test closure is the final stage of the Software Testing Life Cycle (STLC) where all testing-related activities are completed and documented.

*The main objective of the test closure stage is to ensure that all testing-related activities have been completed and that the software is ready for release.

*At the end of the test closure stage, the testing team should have a clear understanding of the software's quality and reliability, and any defects or issues that were identified during testing should have been resolved.

*The test closure stage also includes documenting the testing process and any lessons learned so that they can be used to improve future testing processes

*Test closure is the final stage of the Software Testing Life Cycle (STLC) where all testing-related activities are completed and documented.

*The main activities that take place during the test closure stage include:

*Test summary report: A report is created that summarizes the overall testing process, including the number of test cases executed, the number of defects found, and the overall pass/fail rate.

*Defect tracking: All defects that were identified during testing are tracked and managed until they are resolved.

*Test environment clean-up: The test environment is cleaned up, and all test data and test artifacts are archived.

*Test closure report: A report is created that documents all the testing-related activities that took place, including the testing objectives, scope, schedule, and resources used.

*Knowledge transfer: Knowledge about the software and testing process is shared with the rest of the team and any stakeholders who may need to maintain or support the software in the future.

*Feedback and improvements: Feedback from the testing process is collected and used to improve future testing processes

It is important to note that test closure is not just about documenting the testing process, but also about ensuring that all relevant information is shared and any lessons learned are captured for future reference.

The goal of test closure is to ensure that the software is ready for release and that the testing process has been conducted in an organized and efficient manner.

Conclusion Phase:

*This STLC phase concentrates on the exit criteria and reporting. *Depending on your project and stakeholders' choice, you can decide on reporting whether you want to send out a daily report or the weekly report, etc.

## Software Requirements Specification (SRS)

**********

*A Software Requirements Specification (SRS) is a document that describes what the software will do and how it will be expected to perform.

*It also describes the functionality the product needs to fulfill the needs of all stakeholders (business, users).

* SRS is a formal report, which acts as a representation of software that enables the customers to review whether it (SRS) is according to their requirements.

*Also, it comprises user requirements for a system as well as detailed specifications of the system requirements.

* The SRS is a specification for a specific software product, program, or set of applications that perform particular functions in a specific environment.

*It serves several goals depending on who is writing it.

*First, the SRS could be written by the client of a system.

*Second, the SRS could be written by a developer of the system.

*The two methods create entirely various situations and establish different purposes for the document altogether.

*The first case, SRS, is used to define the needs and expectation of the users.

*The second case, SRS, is written for various purposes and serves as a contract document between customer and developer.

Types of SRS:

1.Functional Requirement

2.Non-Functional Requirement

3.Domain Requirement

1.Functional Requirement:

*In a functional requirement, the user can tell about the working process of the software in an SRS document that the developer must have to fulfill the user's need.

*Working process requirements like how the software will work and how it will execute its aim.

*This is basically known as a "Functional Requirement".

2.Non-Functional Requirement:

*In a non-functional requirement user will tells about the expected characteristics of the software like about security, performance, Storage, Configuration, Cost & software accessibility etc.

*It is known as "a non-functional requirement".

3.Domain Requirement:

*In a domain requirement, it is related to a specific kind of software, purpose or industry vertical.

*It can be the functional and non-functional requirement.

*This type of requirement comes from the domain when the domain falls in the category of system.

*Example: suppose you are running a website where you sell laptops, So the requirement of your site is “Add to Cart” functionality, “Online payment” methods functionality, “Product Listing” functionality and so on.

*Its basically tell if your software fall in a specific category that means there is also need to fulfill the domain requirement by the developer.

*It is basically known as ” Domain Requirement”.

Characteristics of good SRS:

Following are the features of a good SRS document:

1.Correctness:

*User review is used to provide the accuracy of requirements stated in the SRS.

*SRS is said to be perfect if it covers all the needs that are truly expected from the system.

2. Completeness:

*The SRS is complete if, and only if, it includes the following elements:

*All essential requirements, whether relating to functionality, performance, design, constraints, attributes, or external interfaces.

*Definition of their responses of the software to all realizable classes of input data in all available categories of situations.

*Full labels and references to all figures, tables, and diagrams in the SRS and definitions of all terms and units of measure.

3. Consistency:

*The SRS is consistent if, and only if, no subset of individual requirements described in its conflict.

*There are three types of possible conflict in the SRS:

*The specified characteristics of real-world objects may conflicts. For example,

1.The format of an output report may be described in one requirement as tabular but in another as textual.

*One condition may state that all lights shall be green while another states that all lights shall be blue.

2. There may be a reasonable or temporal conflict between the two specified actions. For example,

(a) One requirement may determine that the program will add two inputs, and another may determine that the program will multiply them.

(b) One condition may state that "A" must always follow "B," while other requires that "A and B" co-occurs.

3. Two or more requirements may define the same real-world object but use different terms for that object. For example, a program's request for user input may be called a "prompt" in one requirement's and a "cue" in another.

*The use of standard terminology and descriptions promotes consistency.

4. Unambiguousness:

*SRS is unambiguous when every fixed requirement has only one interpretation.

*This suggests that each element is uniquely interpreted. In case there is a method used with multiple definitions, the requirements report should determine the implications in the SRS so that it is clear and simple to understand.

5. Ranking for importance and stability:

*The SRS is ranked for importance and stability if each requirement in it has an identifier to indicate either the significance or stability of that particular requirement.

*Typically, all requirements are not equally important.

*Some prerequisites may be essential, especially for life-critical applications, while others may be desirable.

*Each element should be identified to make these differences clear and explicit.

*Another way to rank requirements is to distinguish classes of items as essential, conditional, and optional.

6. Modifiability:

*SRS should be made as modifiable as likely and should be capable of quickly obtain changes to the system to some extent.

*Modifications should be perfectly indexed and cross-referenced.

7. Verifiability:

*SRS is correct when the specified requirements can be verified with a cost-effective system to check whether the final software meets those requirements.

*The requirements are verified with the help of reviews.

8. Traceability:

The SRS is traceable if the origin of each of the requirements is clear and if it facilitates the referencing of each condition in future development or enhancement documentation.

There are two types of Traceability:

1. Backward Traceability: This depends upon each requirement explicitly referencing its source in earlier documents.

2. Forward Traceability:

*This depends upon each element in the SRS having a unique name or reference number.

*The forward traceability of the SRS is especially crucial when the software product enters the operation and maintenance phase.

*As code and design document is modified, it is necessary to be able to ascertain the complete set of requirements that may be concerned by those modifications.

9. Design Independence:

*There should be an option to select from multiple design alternatives for the final system.

*More specifically, the SRS should not contain any implementation details.

10. Testability:

An SRS should be written in such a method that it is simple to generate test cases and test plans from the report.

11. Understandable by the customer:

*An end user may be an expert in his/her explicit domain but might not be trained in computer science.

*Hence, the purpose of formal notations and symbols should be avoided too as much extent as possible.

*The language should be kept simple and clear.

12. The right level of abstraction:

*If the SRS is written for the requirements stage, the details should be explained explicitly.

*Whereas, for a feasibility study, fewer analysis can be used.

*Hence, the level of abstraction modifies according to the objective of the SRS.

*Properties of a good SRS document.

The essential properties of a good SRS document are the following:

Concise:

*The SRS report should be concise and at the same time, unambiguous, consistent, and complete.

*Verbose and irrelevant descriptions decrease readability and also increase error possibilities.

Structured:

*It should be well-structured.

*A well-structured document is simple to understand and modify.

*In practice, the SRS document undergoes several revisions to cope up with the user requirements.

*Often, user requirements evolve over a period of time.

*Therefore, to make the modifications to the SRS document easy, it is vital to make the report well-structured.

Black-box view:

*It should only define what the system should do and refrain from stating how to do these.

*This means that the SRS document should define the external behavior of the system and not discuss the implementation issues.

*The SRS report should view the system to be developed as a black box and should define the externally visible behavior of the system.

*For this reason, the SRS report is also known as the black-box specification of a system.

Conceptual integrity:

*It should show conceptual integrity so that the reader can merely understand it.

*Response to undesired events: It should characterize acceptable responses to unwanted events.

*These are called system response to exceptional conditions.

Verifiable:

*All requirements of the system, as documented in the SRS document, should be correct.

*This means that it should be possible to decide whether or not requirements have been met in an implementation.

## Levels Of Testing

**********

The levels of software testing involve the different methodologies, which can be used while we are performing the software testing.

In software testing, we have four different levels of testing, which are as discussed below:

Unit Testing

Integration Testing

System Testing

Acceptance Testing

Level1: Unit Testing:

*Unit testing is the first level of software testing, which is used to test if software modules are satisfying the given requirement or not.

*The first level of testing involves analyzing each unit or an individual component of the software application.

*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.

Real Time Example for Unit Testing:

a checking that your car door can be unlocked, where you test that the door is unlocked using your car key, but it is not unlocked using your house key, garage door remote, or your neighbor’s (who happen to have the same car as you) key.

Level2: Integration Testing:

*The second level of software testing is the integration testing.

*It is mainly used to test the data flow from one module or component to other modules.

*In integration testing, the test engineer tests the units or separate components or modules of the software in a group.

*The primary purpose of executing the integration testing is to identify the defects at the interaction between integrated components or units.

*When each component or module works separately, we need to check the data flow between the dependent modules, and this process is known as integration testing.

*We only go for the integration testing when the functional testing has been completed successfully on each application module.

*In simple words, we can say that integration testing aims to evaluate the accuracy of communication among all the modules.

Real Time Example for Integration Testing:

the fuel system may be tested in collaboration with an exhaust system, and later, these two module's working is tested in collaboration with the working of an engine.

Level3: System Testing:

*The third level of software testing is System Testing, which is used to test the software's functional and non-functional requirements.

*It is end-to-end testing where the testing environment is parallel to the production environment. In the third level of software testing, we will test the application as a whole system.

*To check the end-to-end flow of an application or the software as a user is known as System testing.

* System testing is a sequence of different types of tests to implement and examine the entire working of an integrated software computer system against requirements.

Real Time Example for System Testing:

Each component of the automobile, such as the seats, steering, mirror, brake, cable, engine, car structure, and wheels, is made independently. After each item is manufactured, it is tested separately to see whether it functions as intended.

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.

*The software has passed through three testing levels (Unit Testing, Integration Testing, System Testing). Some minor errors can still be identified when the end-user uses the system in the actual scenario.

*Acceptance testing is the squeezing of all the testing processes that are previously done.

*The acceptance testing is also known as User acceptance testing (UAT) and is done by the customer before accepting the final product.

*Usually, UAT is done by the domain expert (customer) for their satisfaction and checks whether the application is working according to given business scenarios and real-time scenarios.

Real Time Example for Acceptance Testing:

acceptance testing is performed to ensure the pen is ready for delivery to end users (students, poets, hobbyists, etc.). Paytm wants to make applications for its customers, and TCS is given the contract to make them.

## GREY BOX TESTING

**********

Definition:

Grey Box 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.

Reasons for using Grey Box Testing:

It provides combined benefits of both Black Box testing and White Box 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.

Steps to perform Grey box Testing:

First, select and identify inputs from Black Box and White Box testing inputs.

Second, Identify expected outputs from these selected inputs.

Third, identify all the major paths to traverse through during the testing period.

The fourth task is to identify sub-functions which are the part of main functions to perform deep level testing.

The fifth task is to identify inputs for sub functions.

The sixth task is to identify expected outputs for sub functions.

The seventh task includes executing a test case for Sub functions.

The eighth task includes verification of the correctness of result.

Techniques of Grey box Testing:

1.Matrix Testing

2.Regression Testing

3.Pattern Testing

4.Orthogonal Array Testing or OAT

1.Matrix Testing:

Matrix technique is a method to remove unused and uninitialized variables by identifying used variables from the program.

2.Regression Testing:

Regression testing is used to verify that modification in any part of software has not caused any adverse or unintended side effect in any other part of the software.

During confirmation testing, any defect got fixed, and that part of software started working as intended, but there might be a possibility that fixed defect may have introduced a different defect somewhere else in the software.

So, regression testing takes care of these type of defects by testing strategies like retest risky use cases, retest within a firewall, retest all, etc.

3.Pattern Testing:

Pattern testing is applicable to such type of software that is developed by following the same pattern of previous software.

In these type of software possibility to occur the same type of defects.

Pattern testing determines reasons of the failure so they can be fixed in the next software.

Usually, automated software testing tools are used in Grey Box methodology to conduct the test process.

Stubs and module drivers provided to a tester to relieve from manually code generation.

4.Orthogonal Array Testing or OAT:

The purpose of this testing is to cover maximum code with minimum test cases.

Test cases are designed in a way that can cover maximum code as well as GUI functions with a smaller number of test cases.

# Functional Testing

**********

Definition:

*Functional Testing is basically defined as a type of testing that verifies that each function of the software application works in conformance with the requirement and specification.

* Functional Testing can be manual or automated.

The process of Functional Testing:

*Testers follow the following steps in the functional testing:

*Tester does verification of the requirement specification in the software application.

*After analysis, the requirement specification tester will make a plan.

*After planning the tests, the tester will design the test case.

*After designing the test, case tester will make a document of the traceability matrix.

*The tester will execute the test case design.

*Analysis of the coverage to examine the covered testing area of the application.

*Defect management should do to manage defect resolving.

Type of Functional Testing Techniques:

1.Unit Testing:

*Unit testing is the type of functional testing technique where the individual units or modules of the application are tested.

*It ensures that each module is working correctly.

2.Integration Testing:

In Integration testing, combined individual units are tested as a group and expose the faults in the interaction between the integrated units.

3.Smoke Testing:

Smoke testing is a type of functional testing technique where the basic functionality or feature of the application is tested as it ensures that the most important function works properly.

4.User Acceptance Testing: User acceptance testing is done by the client to certify that the system meets the requirements and works as intended. It is the final phase of testing before the product release.

5.Interface Testing: Interface testing is a type of software testing technique that checks the proper interaction between two different software systems.

6.Usability Testing: Usability testing is done to measure how easy and user-friendly a software application is.

7.System Testing: System testing is a type of software testing that is performed on the complete integrated system to evaluate the compliance of the system with the corresponding requirements.

8.Regression Testing: Regression testing is done to make sure that the code changes should not affect the existing functionality and the features of the application. It concentrates on whether all parts are working or not.

9.Sanity Testing: Sanity testing is a subset of regression testing and is done to make sure that the code changes introduced are working as expected.

White box Testing: White box testing is a type of software testing that allows the tester to verify the internal workings of the software system. This includes analyzing the code, infrastructure, and integrations with the external system.

Black box Testing: Black box testing is a type of software testing where the functionality of the software system is tested without looking at the internal working or structures of the software system.

Database Testing: Database testing is a type of software testing that checks the schema, tables, etc of the database under test.

Adhoc Testing: Adhoc testing also known as monkey testing or random testing is a type of software testing that does not follow any documentation or test plan to perform testing.

Recovery Testing: Recovery testing is a type of software testing that verifies the software’s ability to recover from the failures like hardware failures, software failures, crashes, etc.

Static Testing: Static testing is a type of software testing which is performed to check the defects in software without actually executing the code of the software application.

Greybox Testing: Grey box testing is a type of software testing that includes black box and white box testing.

Component Testing: Component testing also known as program testing or module testing is a type of software testing that is done after the unit testing. In this, the test objects can be tested independently as a component without integrating with other components.

Functional Testing vs Non-Functional Testing:

The Differences between Functional Testing and Non-Functional Testing:

Parameters

Functional Testing

Non-functional Testing

Definition:

Functional testing verifies the operations and actions of an application.

Non-functional verifies the behavior of an application.

Testing based on:

It is based on the requirements of the customer.

It is based on the expectations of the customer.

Objective:

The objective is to validate software actions.

The objective is to performance of the software system.

Requirements:

Functional testing is carried out using the functional specification.

Non-functional testing is carried out using the performance specifications.

Functionality

It describes what the product does.

It describes how the product works.

Example:

1.Unit testing.

2.Integration testing.

3.Sanity testing

4.Smoke testing.

5.Regression testing.

1.Performance testing.

2.Load testing.

3.Stress testing.

4.Volume testing.

5.Usability testing.

## Bug Life Cycle

**********

What is bug life cycle in software testing?

*A bug life cycle in software testing is a set of statuses designed to coordinate defect management.

*A bug status helps keep all the members of the development team posted on the progress.

*The cycle starts when a QA engineer reports a new issue found in the tested software and finishes when this issue is sold.

* The Defect Life Cycle, also known as the Bug Life Cycle.

Defect States:

1.New

2.Assigned

3.Open

4.Fixed

5.Pending

6.Retest

7.Reopen

8.Verified

9.Closed

The below four states namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason.

## Black Box Testing

**********

*The Black Box Test is a test that only considers the External Behavior of the system.

*The internal workings of the software is not taken into account.

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

Steps of Black Box Testing:

*The black box test is based on the specification of requirements, so it is examined in the beginning.

*In the second step, the tester creates a positive test scenario and an adverse test scenario by selecting valid and invalid input values to check that the software is processing them correctly or incorrectly.

*In the third step, the tester develops various test cases such as decision table, all pairs test, equivalent division, error estimation, cause-effect graph, etc.

*The fourth phase includes the execution of all test cases.

*In the fifth step, the tester compares the expected output against the actual output.

*In the sixth and final step, if there is any flaw in the software, then it is cured and tested again.

Techniques that are Used in Black Box Testing:

1.Boundary Value Technique

2.Equivalence partitioning Technique

3.Decision Table Technique

4.State Transition Technique

5.All-pair testing Technique

6.Use case Technique

7.Error guessing Technique

8.Cause-Effect Technique

1.Boundary Value Technique:

*BVA is another Black Box Test Design Technique, which is used to find the errors at boundaries of input domain (tests the behavior of a program at the input boundaries) rather than finding those errors in the centre of input.

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

Example:

Assume that, age is a variable of any function, and its minimum value is 18 and the maximum value is 30, both 18 and 30 will be considered as boundary values.

2.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.

*In Other words, Equivalence partitioning is a technique that divides the input domain of a system into partitions or classes that are expected to produce the same output or behavior.

Example: If a system accepts an integer between 1 and 100 as input, you can create four partitions: 1-10, 11-50, 51-99, and 100.

How we perform equivalence partitioning:

Condition1:

*If the requirement is a range of values, then derive the test case for one valid and two invalid inputs.

*Here, the Range of values implies that whenever we want to identify the range values, we go for equivalence partitioning to achieve the minimum test coverage. And after that, we go for error guessing to achieve maximum test coverage.

According to pressman:

*For example, the Amount of test field accepts a Range (100-400) of values:

Condition2:

If the requirement is a set of values, then derive the test case for one valid and two invalid inputs.

Here, Set of values implies that whenever we have to test a set of values, we go for one positive and two negative inputs, then we moved for error guessing, and we also need to verify that all the sets of values are as per the requirement.

Example 1:

Based on the Pressman Method:

If the Amount Transfer is (100000-700000)

Then for, 1 lakh →Accept

And according to General Practice method:

The Range + Percentage given to 1 lakh - 7 lakh

Like: 1lak - 3lak →5.60%

3lak - 6lak →3.66%

6lak - 7lak →Free

Advantages and disadvantages of Equivalence Partitioning technique:

It is process-oriented

All necessary inputs may not cover.

We can achieve the Minimum test coverage

This technique will not consider the condition for boundary value analysis.

It helps to decrease the general test execution time and also reduce the set of test data.

The test engineer might assume that the output for all data set is right, which leads to the problem during the testing process.