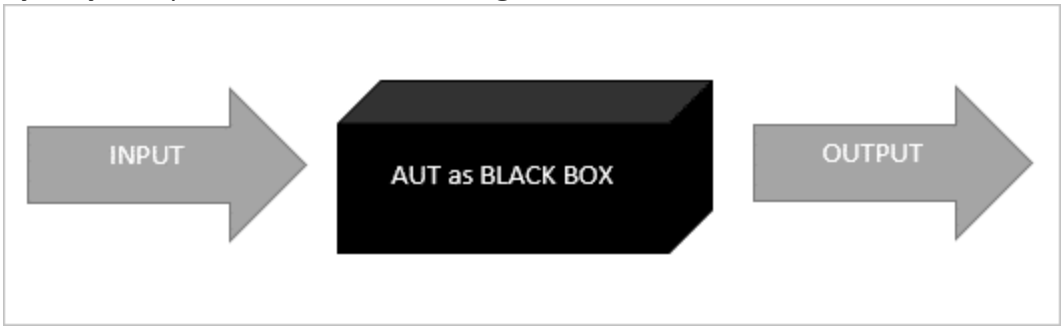
What Is Black Box Testing?
ISTQB Definition – Black Box Testing: Testing an Application Under Test (AUT) without referencing the internal structure is called the black box testing. Testing will be done by visualizing the application as a black box.
Black Box Test Technique: A testing technique to derive the test cases based on the functionality of the application and not considering the internal structure of the system.
Synonyms: Specification-Based Testing



Black box testing is a testing approach that is used to test the functionality of the AUT based on the specifications/SRS without any knowledge of the technology used to implement the application under test.

In the black-box testing, major testing will be around possible inputs and expected outputs. A tester should be able to choose the valid test data carefully. In simple terms, a tester can only see the actions of the AUT. The tester need not know how those actions are performed.

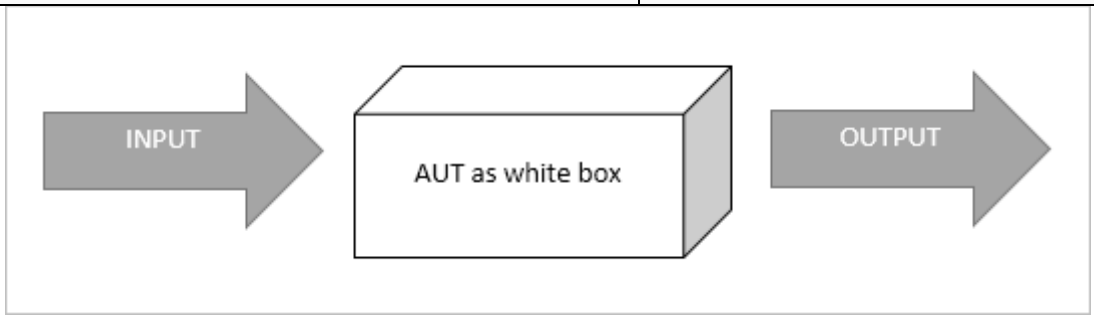
Example: A simple example of black-box testing is a TV (Television). As a user, we watch the TV but we don't need the knowledge of how the TV is built and how it works, etc. We just need to know how to operate the remote control to switch on, switch off, change channels, increase/decrease volume, etc.
In this example,

The **TV** is your **AUT (Application Under Test).**
The **remote control** is the User Interface (UI) that you use to test.
You just need to know how to use the application.

What Is White Box Testing?
ISTQB Definition – White Box Testing: Testing an application with reference to the internal structure of the software component is called white box testing.
White-box test technique: A Procedure to derive and/or select test cases based on an analysis of the internal structure of a component or system.
Synonyms: Clear-box testing, Code-based testing, Glass-box testing, Logic-coverage testing, Logic-driven testing, Structural testing, Structure-based testing, etc.

S.No	Black Box Testing	White Box Testing
1	The main objective of this testing is to test the Functionality / Behavior of the application.	The main objective is to test the infrastructure of the application.
2	This can be performed by a tester without any coding knowledge of the AUT (Application Under Test).	Tester should have the knowledge of internal structure and how it works.
3	Testing can be performed only using the GUI.	Testing can be done at an early stage before the GUI gets ready.
4	This testing cannot cover all possible inputs.	This testing is more thorough as it can test each path.
5	Some test techniques include Boundary Value Analysis, Equivalence Partitioning, Error Guessing etc.	Some testing techniques include Conditional Testing, Data Flow Testing, Loop Testing etc.
6	Test cases should be written based on the Requirement Specification.	Test cases should be written based on the Detailed Design Document.
7	Test cases will have more details about input conditions, test steps, expected results and test data.	Test cases will be simple with the details of the technical concepts like statements, code coverage etc.
8	This is performed by professional Software Testers.	This is the responsibility of the Software Developers.
9	Programming and implementation knowledge is not required.	Programming and implementation knowledge is required.
10	Mainly used in higher level testing like Acceptance Testing, System Testing etc.	Is mainly used in the lower levels of testing like Unit Testing and Integration Testing.
11	This is less time consuming and exhaustive.	This is more time consuming and exhaustive.
12	Test data will have wide possibilities so it will be tough to identify the correct data.	It is easy to identify the test data as only a specific part of the functionality is focused at a time.
13	Main focus of the tester is on how the application is working.	Main focus will be on how the application is built.
14	Test coverage is less as it cannot create test data for all scenarios.	Almost all the paths/application flow are covered as it is easy to test in parts.
15	Code related errors cannot be identified or technical errors cannot be identified.	Helps to identify the hidden errors and helps in optimizing code.
16	Defects are identified once the basic code is developed.	Early defect detection is possible.

S.No	Black Box Testing	White Box Testing
17	User should be able to identify any missing functionalities as the scope of this testing is wide.	Tester cannot identify the missing functionalities as the scope is limited only to the implemented feature.
18	Code access is not required.	Code access is required.
19	Test coverage will be less as the tester has limited knowledge about the technical aspects.	Test coverage will be more as the testers will have more knowledge about the technical concepts.
20	Professional tester focus is on how the entire application is working.	Tester/Developer focus is to check whether the particular path is working or not.



White box testing is a test approach that is used to test the implementation part of an application under test. To perform this testing, the tester/possibly the developer should know the internal structure of the application and how it works.

Example: A Car mechanic should know the internal structure of the car engine to repair it. In this example,

CAR is the **AUT (Application Under Test).**
The **user** is the **black box tester.**
The **mechanic** is the **white box tester.**
These are the basic definitions of white and black box testing and each test method has different techniques to follow.

Difference Between Black Box And White Box Testing

Conclusion

White box and black box testing are necessary for the successful software delivery but 100% testing is not possible in either of the cases.

The major responsibility of the tester is to identify the relevant test types and techniques for a specific application which will result in finding maximum defects and thereby improving the efficiency of the application.

A tester should be able to identify how much testing can be done either in the black box or in the white box testing to certify that an application is working as expected.

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

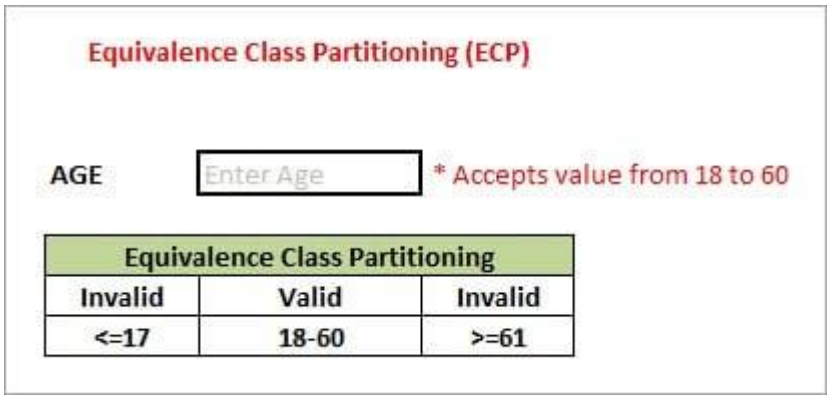
Let's understand each technique in detail.

#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:



Equivalence Class Partitioning		
Invalid	Valid	Invalid
<=17	18-60	>=61

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.

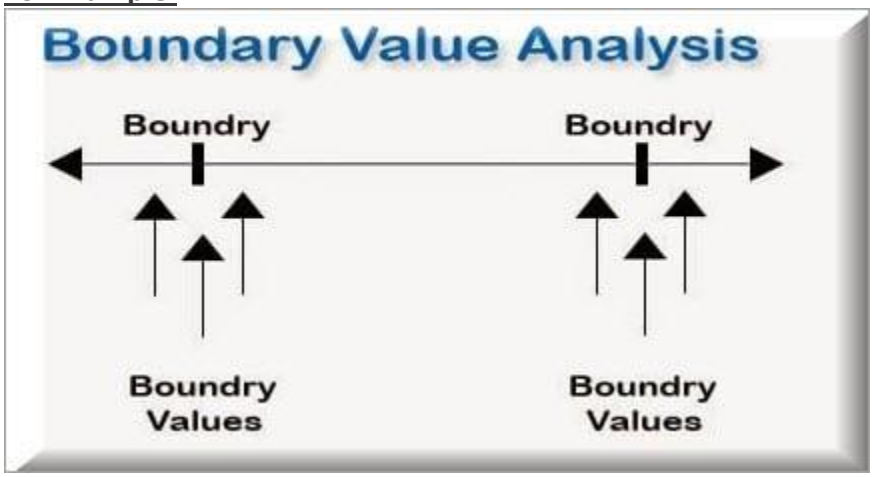
Recommended Read => What is Equivalence Partitioning?

#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:



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.

Decision Table / Cause-Effect

Decision Table	Rule 1	Rule 2	Rule 3	Rule 4
Conditions				
C1 - Male	F	F	T	T
C2 - Senior Citizen	F	T	F	T
Actions				
A1 - Interest Rate 10%				X
A2 - Interest Rate 9%	X	X	X	

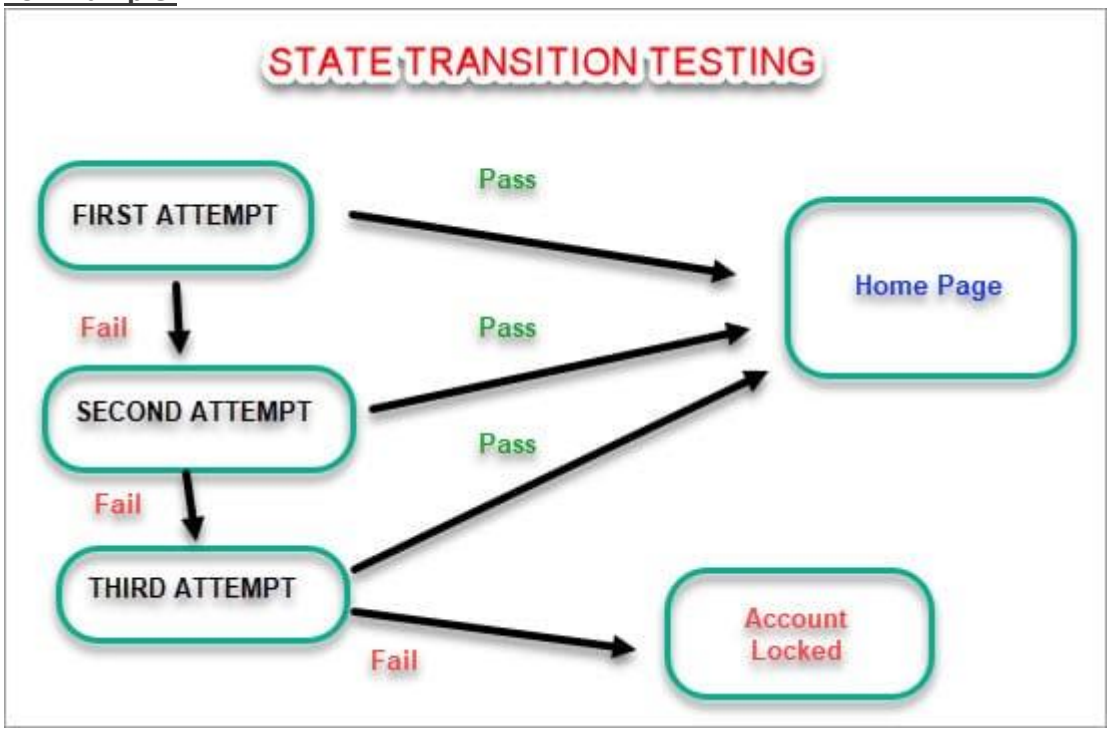
In this example condition, C1 has two values as true and false, C2 also has two values as true and false. The total number of possible combinations would then be four. This way we can derive test cases using a decision table.

#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:



#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 is coverage of the specification in the code:

1. Code coverage

2. Segment coverage: Ensure that each code statement is executed once.

3. Branch Coverage or Node Testing: Coverage of each code branch in from all possible was.

4. Compound Condition Coverage: For multiple conditions test each condition with multiple paths and combination of the different path to reach that condition.

5. Basis Path Testing: Each independent path in the code is taken for testing.

6. Data Flow Testing (DFT): In this approach you track the specific variables through each possible calculation, thus defining the set of intermediate paths through the code.DFT tends to reflect dependencies but it is mainly through sequences of data manipulation. In short, each data variable is tracked and its use is verified. This approach tends to uncover bugs like variables used but not initialize, or declared but not used, and so on.

7. Path Testing: Path testing is where all possible paths through the code are defined and covered. It's a time-consuming task.

8. Loop Testing: These strategies relate to testing single loops, concatenated loops, and nested loops. Independent and dependent code loops and values are tested by this approach.

Why we perform WBT?

To ensure:

- That all independent paths within a module have been exercised at least once.
- All logical decisions verified on their true and false values.
- All loops executed at their boundaries and within their operational bounds internal data structures validity.

To discover the following types of bugs:

- Logical error tend to creep into our work when we design and implement functions, conditions or controls that are out of the program
- The design errors due to difference between logical flow of the program and the actual implementation
- Typographical errors and syntax checking

Does this testing requires detailed programming skills?
We need to write test cases that ensure the complete coverage of the program logic.
For this we need to know the program well i.e. We should know the specification and the code to be tested. Knowledge of programming languages and logic is required for this type of testing.

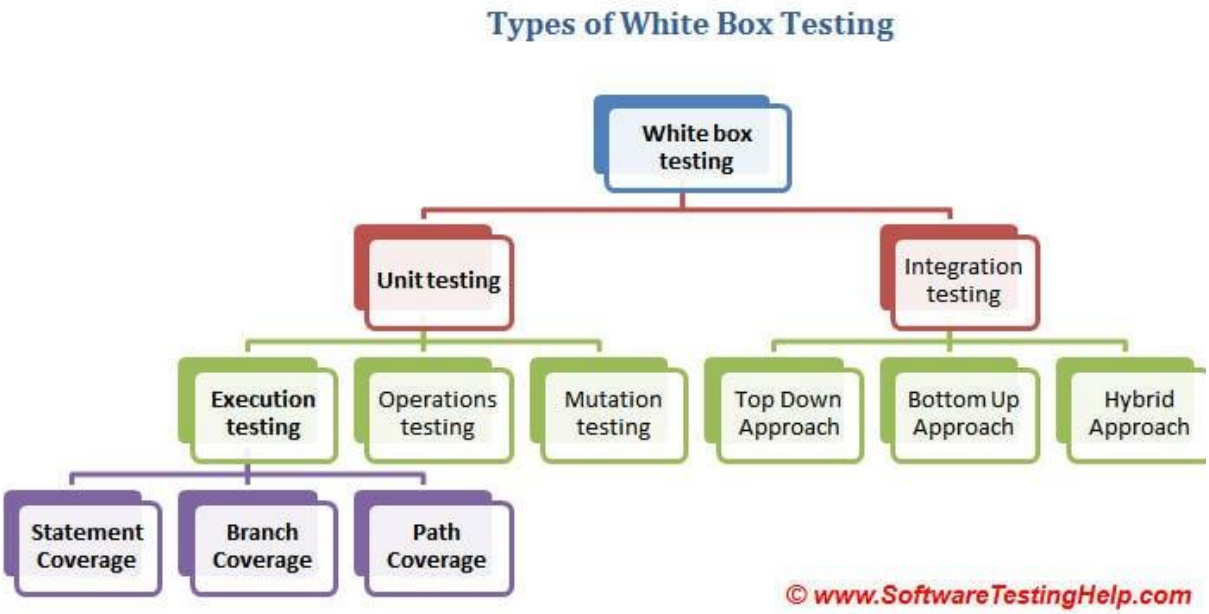
Limitations
Not possible for testing each and every path of the loops in the program. This means exhaustive testing is impossible for large systems.

This does not mean that WBT is not effective. By selecting important logical paths and data structure for testing is practically possible and effective.

Steps to Perform WBT
Step #1 – Understand the functionality of an application through its source code. Which means that a tester must be well versed with the programming language and the other tools as well techniques used to develop the software.
Step #2– Create the tests and execute them.
When we discuss the concept of testing, "coverage" is considered to be the most important factor. Here I will explain how to have maximum coverage from the context of White box testing.

Types and Techniques of White Box Testing
There are several types and different methods for each white box testing type.

See the below image for your reference.



Today, we are going to focus mainly on the ***execution testing types of 'Unit testing white box technique'.***

3 Main White Box Testing Techniques:

1. Statement Coverage
2. Branch Coverage
3. Path Coverage

Note that the statement, branch or path coverage does not identify any bug or defect that needs to be fixed. It only identifies those lines of code which are either never executed or remains untouched. Based on this further testing can be focused on.

Let's understand these techniques one by one with a simple example.

#1) Statement coverage:

In a programming language, a statement is nothing but the line of code or instruction for the computer to understand and act accordingly. A statement becomes an executable statement when it gets compiled and converted into the object code and performs the action when the program is in a running mode.

Hence *"Statement Coverage"*, as the name itself suggests, it is the method of validating whether each and every line of the code is executed at least once.

#2) Branch Coverage:

"Branch" in a programming language is like the "IF statements". An IF statement has two branches: T**rue and False**.

So in Branch coverage (also called Decision coverage), we validate whether each branch is executed at least once.

In case of an "IF statement", there will be two test conditions:

- One to validate the true branch and,
- Other to validate the false branch.

Hence, in theory, Branch Coverage is a testing method which is when executed ensures that each and every branch from each decision point is executed.

#3) Path Coverage

Path coverage tests all the paths of the program. This is a comprehensive technique which ensures that all the paths of the program are traversed at least once. Path Coverage is even more powerful than Branch coverage. This technique is useful for testing the complex programs.

White Box Testing Example

Consider the below simple pseudocode:

INPUT A & B

C = A + B

IF C>100

PRINT "ITS DONE"

For ***Statement Coverage*** – we would only need one test case to check all the lines of the code.

That means:

If I consider *TestCase_01 to be (A=40 and B=70),* then all the lines of code will be executed.

Now the question arises:

1. Is that sufficient?
2. What if I consider my Test case as A=33 and B=45?

Because Statement coverage will only cover the true side, for the pseudo code, only one test case would NOT be sufficient to test it. As a tester, we have to consider the negative cases as well.

Hence for maximum coverage, we need to consider **"*Branch Coverage*"**, which will evaluate the "FALSE" conditions.

In the real world, you may add appropriate statements when the condition fails.

So now the pseudocode becomes:

INPUT A & B

C = A + B

IF C>100

PRINT "ITS DONE"

ELSE

PRINT “ITS PENDING”

Since Statement coverage is not sufficient to test the entire pseudo code, we would require Branch coverage to ensure maximum coverage**.**
So for Branch coverage, we would require two test cases to complete the testing of this pseudo code.

TestCase_01: A=33, B=45
TestCase_02: A=25, B=30
With this, we can see that each and every line of the code is executed at least once.

Here are the Conclusions that are derived so far:

- Branch Coverage ensures more coverage than Statement coverage.
- Branch coverage is more powerful than Statement coverage.
- 100% Branch coverage itself means 100% statement coverage.
- But 100 % statement coverage does not guarantee 100% branch coverage.

Now let’s move on to ***Path Coverage:***

As said earlier, Path coverage is used to test the complex code snippets, which basically involve loop statements or combination of loops and decision statements.

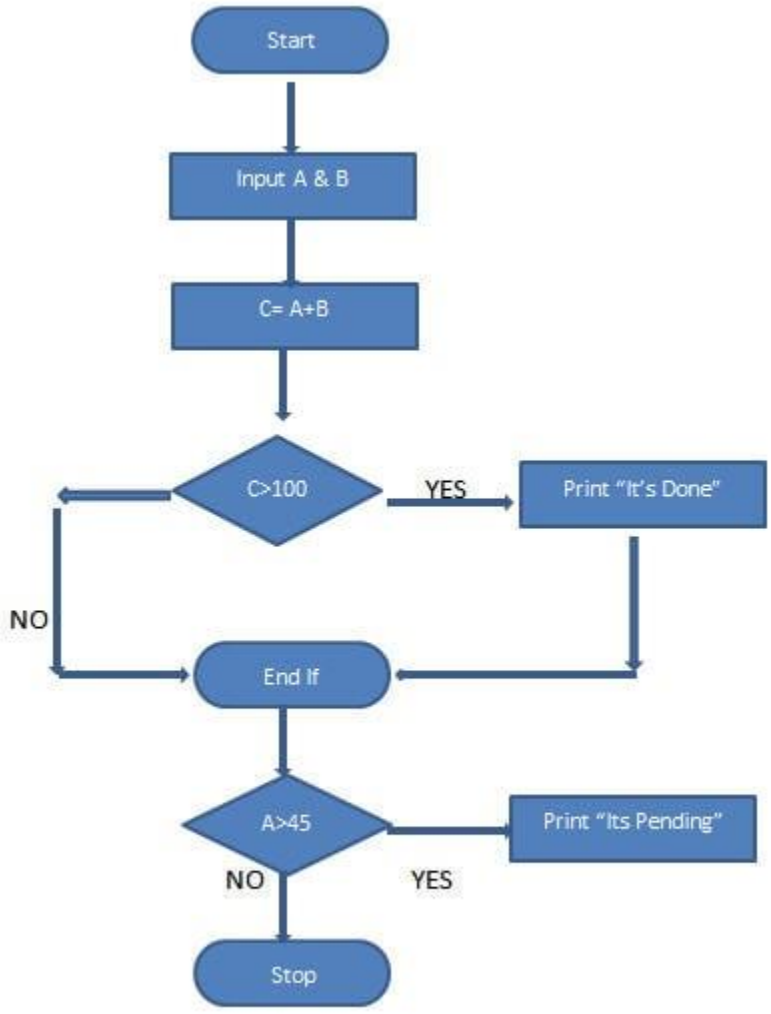
Consider this pseudocode:
INPUT A & B

```
C = A + B

IF C>100

PRINT “ITS DONE”

END IF

IF A>50

PRINT “ITS PENDING”

END IF
```

Now to ensure maximum coverage, we would require 4 test cases.

How? Simply – there are 2 decision statements, so for each decision statement, we would need two branches to test. One for true and the other for the false condition. So for 2 decision statements, we would require 2 test cases to test the true side and 2 test cases to test the false side, which makes a total of 4 test cases.

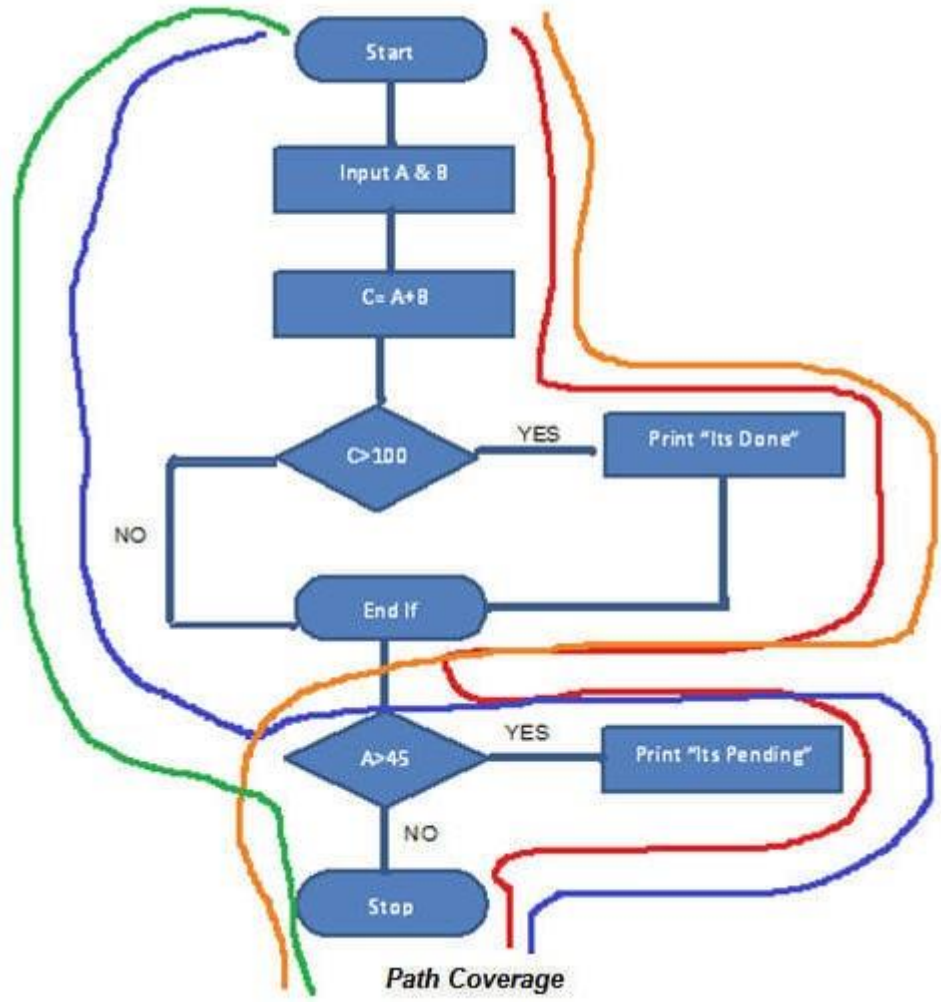
To simplify these let’s consider below flowchart of the pseudo code we have:



Path Coverage

In order to have the full coverage, we would need following test cases:

TestCase_01: A=50, B=60
TestCase_02: A=55, B=40
TestCase_03: A=40, B=65
TestCase_04: A=30, B=30
So the path covered will be:



Red Line – TestCase_01 = (A=50, B=60)

Blue Line = TestCase_02 = (A=55, B=40)

Orange Line = TestCase_03 = (A=40, B=65)

Green Line = TestCase_04 = (A=30, B=30)

****

=>> **Contact us** to suggest your listing here

****

White Box Testing Tools

Given below is a list of top white box test tools.

#1) Veracode

Veracode's white box testing tools will help you in identifying and resolving the software flaws quickly and easily at a reduced cost. It supports several application languages like

.NET, C++, JAVA etc and also enables you to test the security of desktop, web as well as mobile applications. Still, there are several other benefits of Veracode tool. For detailed information about Veracode White box test tools, please check the below link.

#2) EclEmma

EclEmma was initially designed for test runs and analysis within the Eclipse workbench. It is considered to be a free Java code coverage tool and has several features as well. To install or know more about EclEmma please check out the below link.

#3)RCUNIT

A framework which is used for testing C programs is known as RCUNIT. RCUNIT can be used accordingly based on the terms of the MIT License. It is free to use and in order to install or know more about it, please check the below link.

#4) cfix

cfix is one of the unit testing frameworks for C/C++ which solely aims at making test suites development as simple and easy as possible. Meanwhile, cfix is typically specialized for NT Kernel mode and Win32. To install and know more about cfix, please check out the below link

#5) Googletest

Googletest is Google's C++ test framework. Test Discovery, Death tests, Value-parameterized tests, fatal & non-fatal failures, XML test report generation etc are few features of GoogleTest but there are several other features too. Linux, Windows, Symbian, Mac OS X are few platforms where GoogleTest has been used. In order to Download, please check the below link.

#6) EMMA

Emma is an easy to use free JAVA code coverage tool. It includes several features and benefits. To Download and know more about Emma, please check the below link.

Download Link: EMMA

#7) NUnit

NUnit is an easy to use open source unit testing framework which does not require any manual intervention to judge the test results. It supports all .NET languages. It also supports data-driven tests and tests run parallel under NUnit. Earlier releases of NUnit used NUnit license but NUnit 3 is released under the MIT license. But both the licenses allow free use without any restrictions. In order to download and know more about NUnit please check the below link.

#8) CppUnit

CppUnit is a unit testing framework written in C++ and is considered to be the port of JUnit. The test output for CppUnit may be either in the XML or text format. It creates unit tests with its own class and runs tests in the test suites. It is licensed under LGPL. In order to download and know more about CppUnit please check the below link.

#9) JUnit

JUnit is a quiet simple unit testing framework that supports test automation in Java Programming Language. It mainly supports in Test Driven Development and provides the Test coverage report as well. It is licensed under Eclipse Public License. For free download and in order to know more about JUnit please check the below link.

Download Link: JUnit

#10) JsUnit

JsUnit is considered to be the port of JUnit to javascript. And it is an open source unit testing framework to support Client sided Javascript. It is licensed under GNU Public License 2.0, GNU Lesser Public License 2.1 and Mozilla Public License 1.1. In order to download and know more about JsUnit please check the below link.

Also, check all the tools that we have listed under ***Static code analysis here****.*
Feel free to suggest more simple or advanced tools that you are using for white box technique.

Conclusion

Relying only on black box testing is not sufficient for maximum test coverage. We need to have a combination of both black box and white box testing techniques to cover maximum defects.

If done properly, White box testing will certainly contribute to the software quality. It's also good for testers to participate in this testing as it can provide the most "unbiased" opinion about the code. :)