**Software Testing Techniques**

Once the software is developed it should be tested in a proper manner before the system is delivered to the user. For this, two techniques that provide systematic guidance for designing tests are used:

[*White Box Testing*](https://ecomputernotes.com/software-engineering/testing-techniques#White_Box_Testing)

* *statement coverage*
* *branch coverage*
* *decision coverage*

*Gray Box testing*

[*Black Box Testing*](https://ecomputernotes.com/software-engineering/testing-techniques#Black_Box_Testing)

*- Equivalence class Partitioning*

*- Boundary Value Analysis*

*- Decision Table Testing*

*- State Transition Testing*

*- Use Case Testing*

*Experience-based Test Techniques*

**White box testing**

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

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

Developers do white box testing. In this, the developer will test every line of the code of the program. The developers perform the White-box testing and then send the application or the software to the testing team, where they will perform the [black box testing](https://www.javatpoint.com/black-box-testing) and verify the application along with the requirements and identify the bugs and send it to the developer.

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

Consider the following piece of code

printme (int a, int b) {

int result = a+ b;

If (result> 0)

Print ("Positive", result)

Else

Print ("Negative", result)

}

The goal of WhiteBox testing in software engineering is to verify all the decision branches, loops, and statements in the code.

To exercise the statements in the above white box testing example, WhiteBox test cases would be

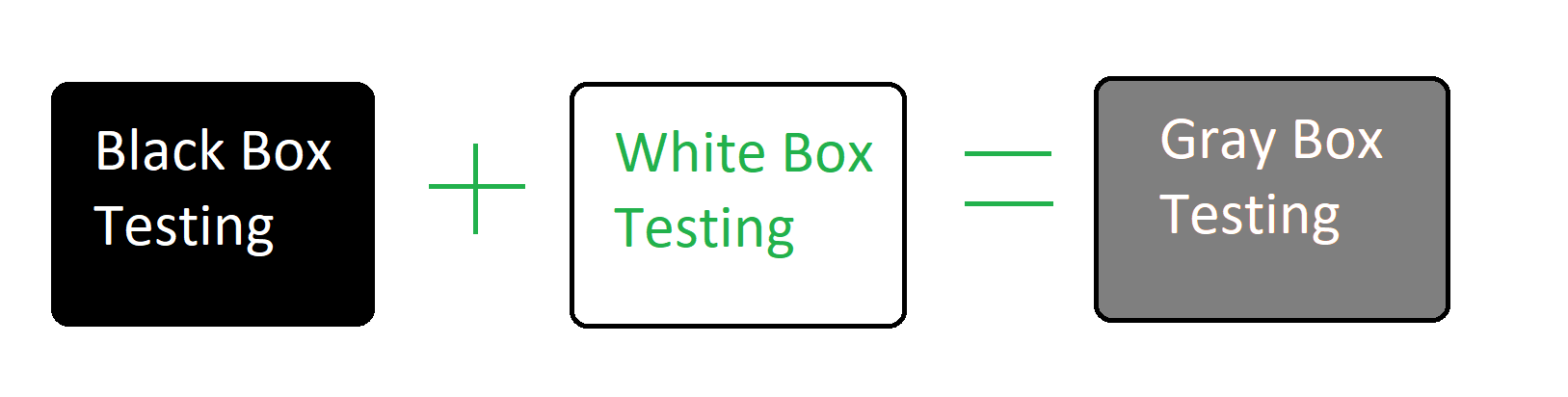
A = 1, B = 1

A = -1, B = -3

**Gray Box Testing**

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

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



**Black box**

Black box (or functional) testing checks the functional requirements and examines the input and output data of these requirements. When black box testing is performed, only the sets of ‘legal’ input and corresponding outputs should be known to the tester and not the internal logic of the program to produce that output.

There are 5 important software testing techniques:

-[Equivalence Class Partitioning](https://www.guru99.com/software-testing-techniques.html#2)

* [Boundary Value Analysis (BVA)](https://www.guru99.com/software-testing-techniques.html#1)
* [Decision Table based testing.](https://www.guru99.com/software-testing-techniques.html#3)
* [State Transition](https://www.guru99.com/software-testing-techniques.html#4)

***Equivalence class partitioning***

An input or output range of values such that only one value in the range becomes a test case

To understand equivalence class partitioning properly, let us consider an example. This example is explained in the series of steps listed below.

Ex 1:

1. Suppose that a program P takes an integer X as input.
2. Now, either X < 0 or X >0.
3. In case X < 0, the program is required to perform task T1; otherwise, the task T2 is performed.
4. The input domain is as large as X and it can assume a large number of values. Therefore the input domain (p) is partitioned into two equivalence classes and all test inputs in the X < 0 and X > 0 equivalence classes are considered to be equivalent.
5. Now, as shown in Figure independent test cases are developed for X < 0 and X > 0.

Ex 2:

Input conditions are valid between:

1 to 10 and 20 to 30

Hence there are five equivalence classes

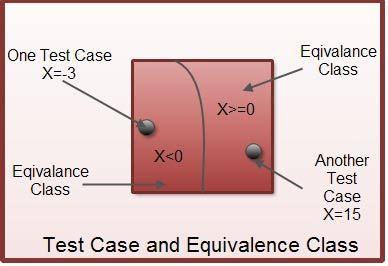
--- to 0 (invalid)

1 to 10 (valid)

11 to 19 (invalid)

20 to 30 (valid)

31 to --- (invalid)



***Boundary value analysis***

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

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

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

Guidelines for Boundary Value analysis

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

Example:

Input condition is valid between 1 to 10

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

***Decision Based Table***

A decision table is also known as a Cause-Effect table. This software testing technique is used for functions which respond to a combination of inputs or events. For example, a submit button should be enabled if the user has entered all required fields.

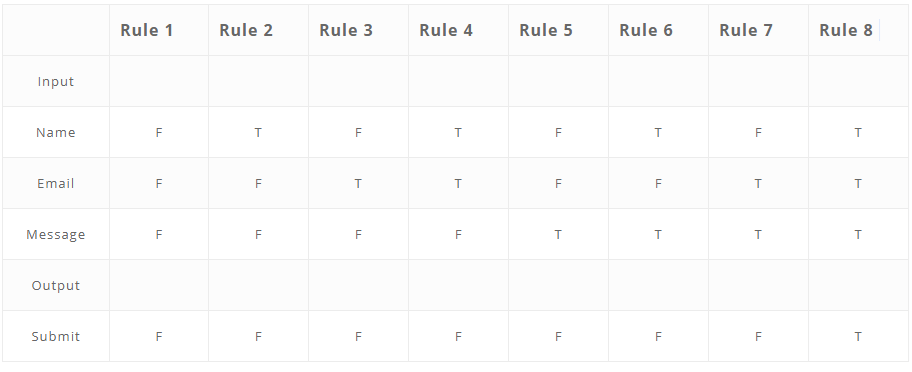
The first task is to identify functionalities where the output depends on a combination of inputs. If there are large input sets of combinations, then divide it into smaller subsets which are helpful for managing a decision table.

For every function, you need to create a table and list down all types of combinations of inputs and its respective outputs. This helps to identify a condition that is overlooked by the tester.

**Following are steps to create a decision table:**

* Enlist the inputs in rows
* Enter all the rules in the column
* Fill the table with the different combination of inputs
* In the last row, note down the output against the input combination.

**Example**: A submit button in a contact form is enabled only when all the inputs are entered by the end user.



***State Transition Method***

In State Transition technique changes in input conditions change the state of the Application Under Test (AUT). This testing technique allows the tester to test the behavior of an AUT. The tester can perform this action by entering various input conditions in a sequence. In State transition technique, the testing team provides positive as well as negative input test values for evaluating the system behavior.

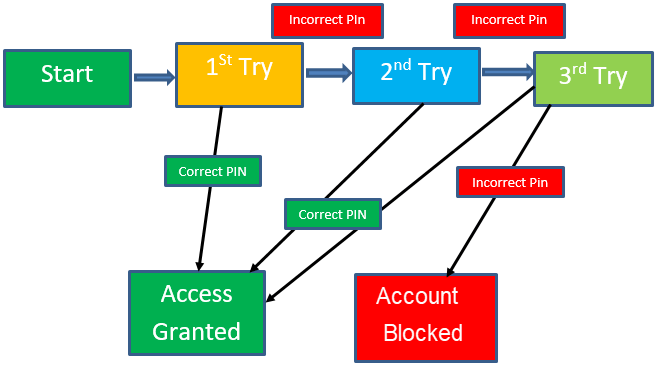
Guideline for State Transition:

* State transition should be used when a testing team is testing the application for a limited set of input values.
* The technique should be used when the testing team wants to test a sequence of events which happen in the application under test.

Example:

In the following example, if the user enters a valid password in any of the first three attempts the user will be able to log in successfully. If the user enters the invalid password in the first or second try, the user will be prompted to re-enter the password. When the user enters password incorrectly 3rd time, the action has taken, and the account will be blocked.

### **State transition diagram:**



***Experience-based Test Techniques***

**Guidelines for Error Guessing:**

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

**Conclusion**

* Software testing Techniques allow you to design better cases. There are five primarily used techniques.
* Boundary value analysis is testing at the boundaries between partitions.
* Equivalent Class Partitioning allows you to divide a set of test conditions into a partition which should be considered the same.
* Decision Table software testing technique is used for functions which respond to a combination of inputs or events.
* In State Transition technique changes in input conditions change the state of the Application Under Test (AUT)
* Error guessing is a software testing technique which is based on guessing the error which can prevail in the code.

## ***Error Guessing***

Error Guessing is a software testing technique based on guessing the error which can prevail in the code. The technique is heavily based on the experience where the test analysts use their experience to guess the problematic part of the testing application. Hence, the test analysts must be skilled and experienced for better error guessing.

The technique counts a list of possible errors or error-prone situations. Then the tester writes a test case to expose those errors. To design test cases based on this software testing technique, the analyst can use the past experiences to identify the conditions.