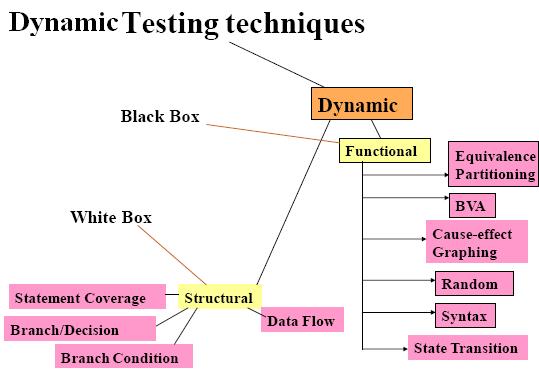
**Notes for MCA-II (Semester- III)**

**Subject :- Software Testing & Quality Assurance**

**(Subject Code:- IT-33)**

**Chapter: 4] Dynamic Testing**

**Dynamic Testing :-**

****

* Under Dynamic Testing code is executed. It checks for functional behavior of software system, memory/cpu usage and overall performance of the system. Hence the name "Dynamic"
* Main objective of this testing is to confirm that the software product works in conformance with the business requirements. This testing is also called as Execution technique or validation testing.
* Dynamic testing executes the software and validates the output with the expected outcome. Dynamic testing is performed at all levels of testing and it can be either black or white box testing.
* **Testing Techniques used for Dynamic Testing:**
* Unit Testing:
* Under unit testing, individual units or modules is tested by the developers. It involves testing of source code by developers.
* Integration Testing:
* Individual modules are grouped together and tested by the developers. The purpose is to determine that modules are working as expected once they are integrated.
* System Testing:
* System testing is performed on the whole system by checking whether the system or application meets the requirement specification document.

|  |  |
| --- | --- |
| **Static Testing** | **Dynamic Testing** |
| Testing done without executing the program | Testing done by executing the program |
| This testing does verification process | Dynamic testing does validation process |
| Static testing is about prevention of defects | Dynamic testing is about finding and fixing the defects |
| Static testing gives assessment of code and documentation | Dynamic testing gives bugs/bottlenecks in the software system. |
| Static testing involves checklist and process to be followed | Dynamic testing involves test cases for execution |
| This testing can be performed before compilation | Dynamic testing is performed after compilation |
| Static testing covers the structural and statement coverage testing | Dynamic testing covers the executable file of the code |
| Cost of finding defects and fixing is less | Cost of finding and fixing defects is high |
| Return on investment will be high as this process involved at early stage | Return on investment will be low as this process involves after the development phase |
| More reviews  comments are highly recommended for good quality | More defects are highly recommended for good quality. |
| Requires loads of meetings | Comparatively requires lesser meetings |

* **4.1 Test Design Techniques-Black Box Testing Techniques:-**

**Black Box Testing** is a software testing method in which the functionalities of software applications are tested without having knowledge of internal code structure, implementation details and internal paths. Black Box Testing mainly focuses on input and output of software applications and it is entirely based on software requirements and specifications. It is also known as Behavioral Testing.



Black box testing is a technique of software testing which examines the functionality of software without peering into its internal structure or coding. The primary source of black box testing is a specification of requirements that is stated by the customer.

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

In this method, tester selects a function and gives input value to examine its functionality, and checks whether the function is giving expected output or not. If the function produces correct output, then it is passed in testing, otherwise failed. The test team reports the result to the development team and then tests the next function. After completing testing of all functions if there are severe problems, then it is given back to the development team for correction.

Black box testing

**Generic 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.
* **4.1.1 Equivalence Partitioning :-**

**Equivalence partitioning or equivalence class partitioning (ECP)** is a [software testing](https://en.wikipedia.org/wiki/Software_testing) technique that divides the input data of a software unit into partitions of equivalent data from which test cases can be derived. In principle, test cases are designed to cover each partition at least once. This technique tries to define test cases that uncover classes of errors, thereby reducing the total number of test cases that must be developed. An advantage of this approach is reduction in the time required for testing software due to lesser number of test cases.

Equivalence partitioning is typically applied to the inputs of a tested component, but may be applied to the outputs in rare cases. The equivalence partitions are usually derived from the requirements specification for input attributes that influence the processing of the test object.

It is used to minimize the number of possible test cases to an optimum level while maintains reasonable test coverage.

It is often seen that many type of inputs work similarly so instead of giving all of them separately we can group them together and test only one input of each group. The idea is to partition the input domain of the system into a number of equivalence classes such that each member of class works in a similar way, i.e., if a test case in one class results in some error, other members of class would also result into same error.

* Equivalence partitioning is a black-box testing method.
* It divides the input domain of a program into classes of data from which test cases can be derived that uncovers classes of errors.
* It is a systematic process that identifies a set of classes of input conditions to be tested.
* An equivalence class represents a set of valid or invalid states of input conditions. Typically, an input condition is either a specific numeric value, a range of values, a set of related values or a Boolean condition.
* The aim is to minimize the number of test cases required to cover these input conditions.
* There are 2 distinct steps. The first is to identify the ***Equivalence Classes (ECs)*** and the second is to identify the test cases.
* For identifying Equivalence classes following are some guidelines that are followed by the tester:

For each external input:

i) If the input specifies a range of valid values, define one valid EC(with in the range) and 2 invalid ECs (one outside each end of the range).

EX: If the input requires a month in the range of 1-12, then define 1 valid EC for months 1 through 12 and 2 invalid ECs (month<1 and month>12).

ii) If the input specifies the number of valid values, define 1 valid EC and 2 invalid Ecs.

EX: If the input requires the titles of atleast 3 but no more than 8 books, then define 1 valid EC and 2 invalid ECs(i.e. <3 and >8books).

iii) If the input specifies a set of valid values, define 1 valid EC(within the set) and one invalid EC(outside the set).

EX: if the input requires one of the names like MCA,MCM,MBA, then define 1 valid EC (using one of the valid names from the above) and 1 invalid EC(using any other name except from the above).

iv) If an input condition is Boolean, 1 valid and 1 invalid classes are defined.

* Thus, equivalence partitioning significantly reduces the number of input conditions to be tested.(It does not test combinations of I/P conditions)

Equivalence partitioning is a technique of software testing in which input data divided into partitions of valid and invalid values, and it is mandatory that all partitions must exhibit the same behavior.

Equivalence Partitioning or Equivalence Class Partitioning is type of black box testing technique which can be applied to all levels of [software testing](https://www.guru99.com/software-testing.html) like unit, integration, system, etc. In this technique, input data units are divided into equivalent partitions that can be used to derive test cases which reduces time required for testing because of small number of test cases.

* It divides the input data of software into different equivalence data classes.

You can apply this technique, where there is a range in the input field.

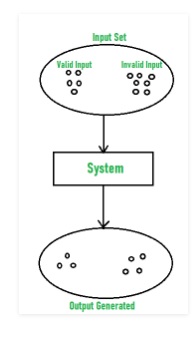
**Guidelines for Equivalence Partitioning : -**

If the range condition is given as an input, then one valid and two invalid equivalence classes are defined.

If a specific value is given as input, then one valid and two invalid equivalence classes are defined.

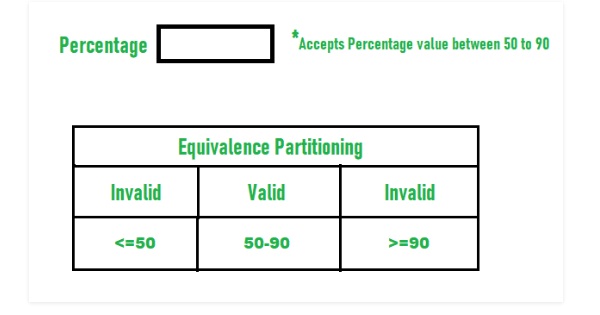
If a member of set is given as an input, then one valid and one invalid equivalence class is defined.

If Boolean no. is given as an input condition, then one valid and one invalid equivalence class is defined.



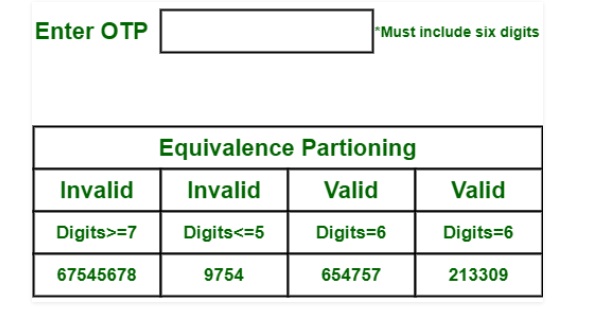
**Example:**   
Let us consider an example of any college admission process. There is a college that gives admissions to students based upon their percentage.

Consider percentage field that will accept percentage only between 50 to 90 %, more and even less than not be accepted, and application will redirect user to an error page. If percentage entered by user is less than 50 %or more than 90 %, that equivalence partitioning method will show an invalid percentage. If percentage entered is between 50 to 90 %, then equivalence partitioning method will show valid percentage.



**Another Example- :**   
Let us consider an example of software application. There is function of software application that accepts only particular number of digits, not even greater or less than that particular number.

Consider an OTP number that contains only 6 digit number, greater and even less than six digits will not be accepted, and the application will redirect customer or user to error page. If password entered by user is less or more than six characters, that equivalence partitioning method will show an invalid OTP. If password entered is exactly six characters, then equivalence partitioning method will show valid OTP.



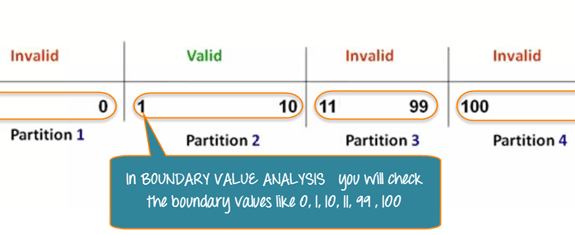
* **4.1.2. Boundary Value Analysis:-**

Boundary testing is the process of testing between extreme ends or boundaries between partitions of the input values.

* So these extreme ends like Start- End, Lower- Upper, Maximum-Minimum, Just Inside-Just Outside values are called boundary values and the testing is called “boundary testing”.
* The basic idea in normal boundary value testing is to select input variable values at their:

1. Minimum
2. Just above the minimum
3. A nominal value
4. Just below the maximum
5. Maximum

**Boundary Value Analysis**- in Boundary Value Analysis, you test boundaries between equivalence partitions



In our earlier equivalence partitioning example, instead of checking one value for each partition, you will check the values at the partitions like 0, 1, 10, 11 and so on. As you may observe, you test values at both valid and invalid boundaries. Boundary Value Analysis is also called range checking.

Equivalence partitioning and boundary value analysis(BVA) are closely related and can be used together at all levels of testing.

Boundary value analysis is one of the widely used case design technique for black box testing. It is used to test boundary values because the input values near the boundary have higher chances of error.

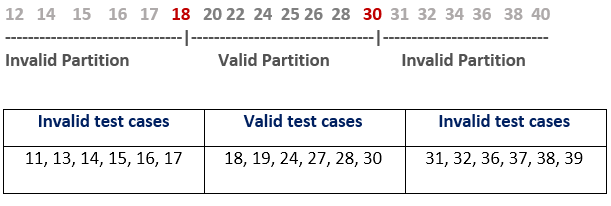
Whenever we do the testing by boundary value analysis, the tester focuses on, while entering boundary value whether the software is producing correct output or not.

Boundary values are those that contain the upper and lower limit of a variable. 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.

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

**Let's understand via practical:**

Imagine, there is a function that accepts a number between 18 to 30, where 18 is the minimum and 30 is the maximum value of valid partition, the other values of this partition are 19, 20, 21, 22, 23, 24, 25, 26, 27, 28 and 29. The invalid partition consists of the numbers which are less than 18 such as 12, 14, 15, 16 and 17, and more than 30 such as 31, 32, 34, 36 and 40. Tester develops test cases for both valid and invalid partitions to capture the behaviour of the system on different input conditions.



The software system will be passed in the test if it accepts a valid number and gives the desired output, if it is not, then it is unsuccessful. In another scenario, the software system should not accept invalid numbers, and if the entered number is invalid, then it should display error massage.

If the software which is under test, follows all the testing guidelines and specifications then it is sent to the releasing team otherwise to the development team to fix the defects.

**Boundary Value Analysis (BVA)** is a black box software testing technique where test cases are designed using boundary values. BVA is based on the single fault assumption, also known as critical fault assumption which states that failures are rarely the product of two or more simultaneous faults. Hence while designing the test cases for BVA we keep all but one variable to the nominal value and allowing the remaining variable to take the extreme value.

**Test Case Design for BVA**:

While designing the test cases for BVA first we determine the number of input variables in the problem. For each input variable, we then determine the range of values it can take. Then we determine the extreme values and nominal value for each input variable.

* **4.1.3 Decision Table Testing :-**

A **Decision Table** is a tabular representation of inputs versus rules/cases/test conditions. It is a very effective tool used for both complex software testing and requirements management. Decision table helps to check all possible combinations of conditions for testing and testers can also identify missed conditions easily. The conditions are indicated as True(T) and False(F) values.

Decision table technique is one of the widely used case design techniques for black box testing. This is a systematic approach where various input combinations and their respective system behaviour are captured in a tabular form.

That's why it is also known as a cause-effect table. This technique is used to pick the test cases in a systematic manner; it saves the testing time and gives good coverage to the testing area of the software application.

Decision table technique is appropriate for the functions that have a logical relationship between two and more than two inputs.

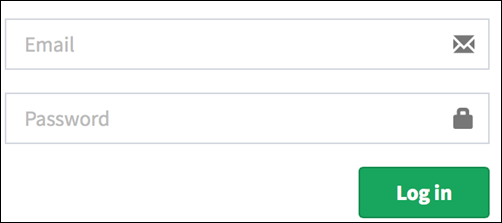
This technique is related to the correct combination of inputs and determines the result of various combinations of input. To design the test cases by decision table technique, we need to consider conditions as input and actions as output.

## What is Decision Table Testing?

Decision table testing is a software testing technique used to test system behaviour for different input combinations. This is a systematic approach where the different input combinations and their corresponding system behaviour (Output) are captured in a tabular form. That is why it is also called as a **Cause-Effect** table where Cause and effects are captured for better test coverage.

## Example 1: How to make Decision Base Table for Login Screen

Let’s create a decision table for a login screen.



The condition is simple if the user provides correct username and password the user will be redirected to the homepage. If any of the input is wrong, an error message will be displayed.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Conditions** | **Rule 1** | **Rule 2** | **Rule 3** | **Rule 4** |
| **Username (T/F)** | F | T | F | T |
| **Password (T/F)** | F | F | T | T |
| **Output (E/H)** | E | E | E | H |

**Legend:**

* T – Correct username/password
* F – Wrong username/password
* E – Error message is displayed
* H – Home screen is displayed

**Interpretation:**

Case 1 – Username and password both were wrong. The user is shown an error message.

Case 2 – Username was correct, but the password was wrong. The user is shown an error message.

Case 3 – Username was wrong, but the password was correct. The user is shown an error message.

Case 4 – Username and password both were correct, and the user navigated to homepage

**While converting this to test case, we can create 2 scenarios** ,

Enter correct username and correct password and click on login, and the expected result will be the user should be navigated to homepage

**And one from the below scenario**

Enter wrong username and wrong password and click on login, and the expected result will be the user should get an error message

Enter correct username and wrong password and click on login, and the expected result will be the user should get an error message

Enter wrong username and correct password and click on login, and the expected result will be the user should get an error message

As they essentially test the same rule.

Why Decision Table Testing is Important?

**Decision Table Testing is Important** because it helps to test different combinations of conditions and provide better test coverage for complex business logic. When testing the behaviour of a large set of inputs where system behaviour differs with each set of input, decision table testing provides good coverage and the representation is simple so it is easy to interpret and use.

In Software Engineering, boundary value and equivalent partition are other similar techniques used to ensure better coverage. They are used if the system shows the **same**behaviour for a large set of inputs. However, in a system where for each set of input values the system behaviour is **different**, boundary value and equivalent partitioning technique are not effective in ensuring good test coverage.

In this case, decision table testing is a good option. This technique can make sure of good coverage, and the representation is simple so that it is easy to interpret and use.

This table can be used as the reference for the requirement and for the functionality development since it is easy to understand and cover all the combinations.

**Advantages of Decision Table Testing**

* When the system behaviour is different for different input and not same for a range of inputs, both equivalent partitioning, and boundary value analysis won’t help, but decision table can be used.
* The representation is simple so that it can be easily interpreted and is used for development and business as well.
* This table will help to make effective combinations and can ensure a better coverage for testing
* Any complex business conditions can be easily turned into decision tables

In a case we are going for 100% coverage typically when the input combinations are low, this technique can ensure the coverage.

**Disadvantages of Decision Table Testing:-**

The main disadvantage is that when the number of input increases the table will become more complex

* **4.1.4 State Transition Testing :-**

**State Transition Testing** is a black box testing technique in which changes made in input conditions cause state changes or output changes in the Application under Test(AUT). State transition testing helps to analyse behaviour of an application for different input conditions. Testers can provide positive and negative input test values and record the system behaviour.

It is the model on which the system and the tests are based. Any system where you get a different output for the same input, depending on what has happened before, is a finite state system.

State Transition Testing Technique is helpful where you need to test different system transitions.

**When to Use State Transition**?

* This can be used when a tester is testing the application for a finite set of input values.
* When the tester is trying to test sequence of events that occur in the application under test. I.e., this will allow the tester to test the application behaviour for a sequence of input values.
* When the system under test has a dependency on the events/values in the past.

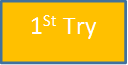
**When to Not Rely On State Transition**?

* When the testing is not done for sequential input combinations.
* If the testing is to be done for different functionalities like exploratory testing

**Four Parts Of State Transition Diagram**

There are 4 main components of the State Transition Model as below

1) **States that the software might get**



**2) Transition** from one state to another

https://www.guru99.com/images/1/103017_0527_WhatIsState2.png

**3) Events** that origin a transition like closing a file or withdrawing money

https://www.guru99.com/images/1/103017_0527_WhatIsState3.png

**4) Actions** that result from a transition (an error message or being given the cash.)



## State Transition Diagram and State Transition Table

There are two main ways to represent or design state transition, State transition diagram, and state transition table.

In state transition diagram the states are shown in boxed texts, and the transition is represented by arrows. It is also called State Chart or Graph. It is useful in identifying valid transitions.

In state transition table all the states are listed on the left side, and the events are described on the top. Each cell in the table represents the state of the system after the event has occurred. It is also called State Table. It is useful in identifying invalid transitions.

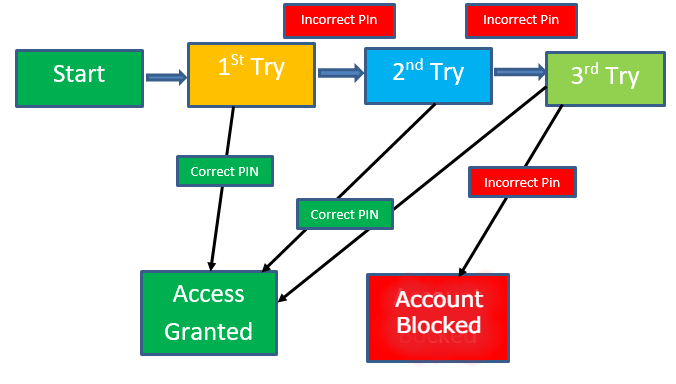
## How to Make a State Transition (Examples of a State Transition)

### Example 1:

Let’s consider an ATM system function where if the user enters the invalid password three times the account will be locked.

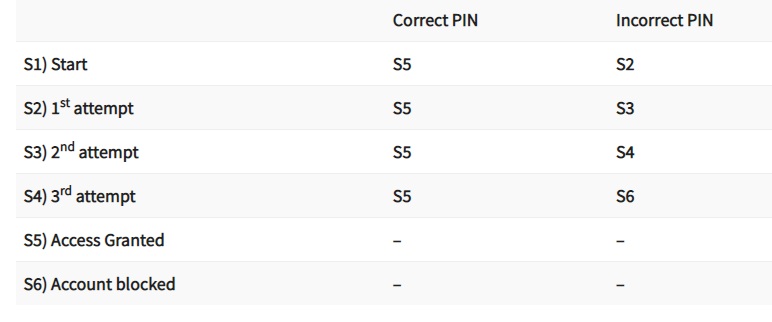
In this system, if the user enters a valid password in any of the first three attempts the user will be logged in successfully. If the user enters the invalid password in the first or second try, the user will be asked to re-enter the password. And finally, if the user enters incorrect password 3rd time, the account will be blocked.

### State transition diagram



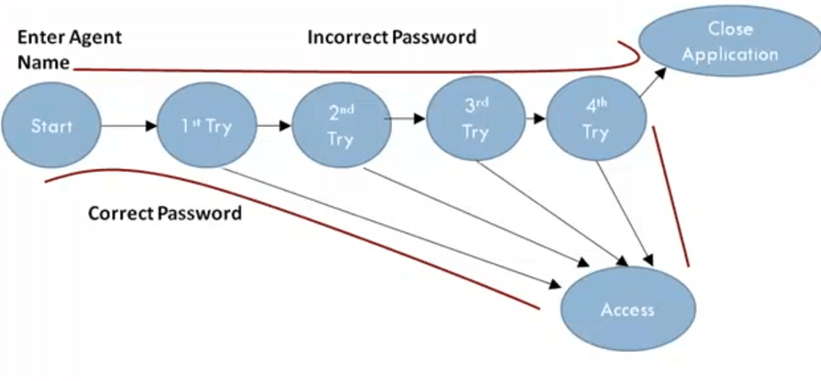
In the diagram whenever the user enters the correct PIN he is moved to Access granted state, and if he enters the wrong password he is moved to next try and if he does the same for the 3rd time the account blocked state is reached.

### State Transition Table



In the table when the user enters the correct PIN, state is transitioned to S5 which is Access granted. And if the user enters a wrong password he is moved to next state. If he does the same 3rd time, he will reach the account blocked state.

In the flight reservation login screen, consider you have to enter correct agent name and password to access the flight reservation application.

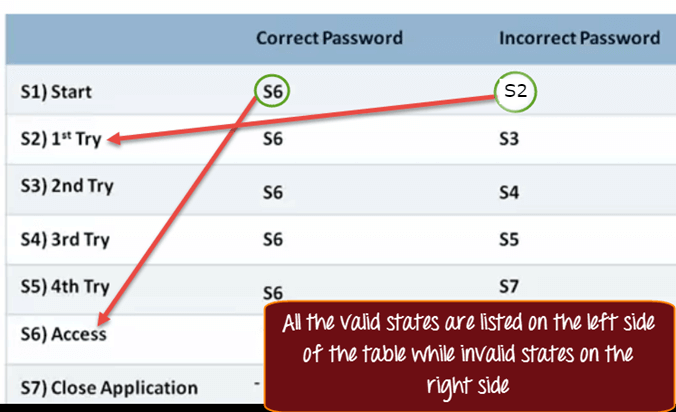


It gives you the access to the application with correct password and login name, but what if you entered the wrong password.

The application allows three attempts, and if users enter the wrong password at 4th attempt, the system closes the application automatically.

The State Graphs helps you determine valid transitions to be tested. In this case, testing with the correct password and with an incorrect password is compulsory. For the test scenarios, log-in on 2nd, 3rd and 4th attempt anyone could be tested.

You can use State Table to determine invalid system transitions.



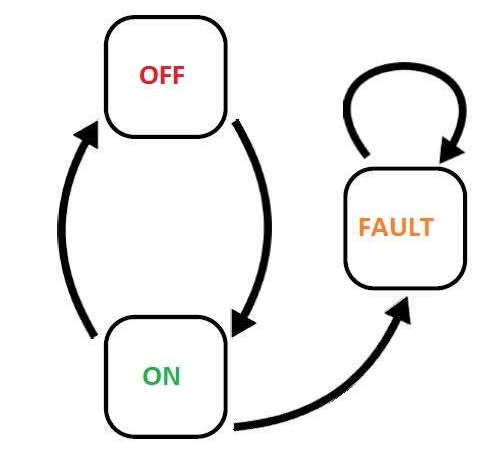
In a State Table, all the valid states are listed on the left side of the table, and the events that cause them on the top.

Each cell represents the state system will move to when the corresponding event occurs.

For example, while in S1 state you enter a correct password you are taken to state S6 (Access Granted). Suppose if you have entered the wrong password at first attempt you will be taken to state S3 or 2nd Try.

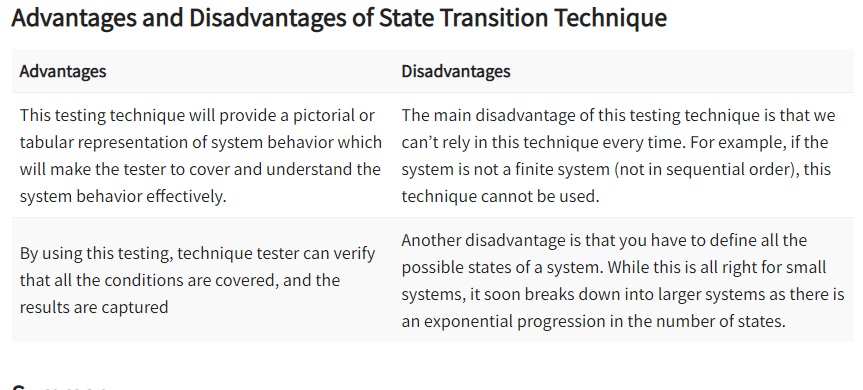
## Example:

A System's transition is represented as shown in the below diagram:



The tests are derived from the above state and transition and below are the possible scenarios that need to be tested.

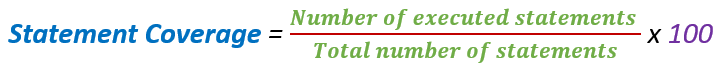
|  |  |  |  |
| --- | --- | --- | --- |
| **Tests** | **Test 1** | **Test 2** | **Test 3** |
| Start State | Off | On | On |
| Input | Switch ON | Switch Off | Switch off |
| Output | Light ON | Light Off | Fault |
| Finish State | ON | OFF | On |

****

* **4.2 Test Design Techniques -White Box Testing Techniques (coverage based and fault-based)**
* **4.2.1. Statement coverage :-**

**Statement Coverage** is a white box testing technique in which all the executable statements in the source code are executed at least once. It is used for calculation of the number of statements in source code which have been executed. The main purpose of Statement Coverage is to cover all the possible paths, lines and statements in source code.

Statement coverage is used to derive scenario based upon the structure of the code under test.



**In**[**White Box Testing**](https://www.guru99.com/white-box-testing.html)**, the tester is concentrating on how the software works. In other words, the tester will be concentrating on the internal working of source code concerning control flow graphs or flow charts.**

**Generally in any software, if we look at the source code, there will be a wide variety of elements like operators, functions, looping, exceptional handlers, etc. Based on the input to the program, some of the code statements may not be executed. The goal of Statement coverage is to cover all the possible path’s, line, and statement in the code.**

**Let’s understand this with an example, how to calculate statement coverage.**

**Scenario to calculate Statement Coverage for given source code. Here we are taking two different scenarios to check the percentage of statement coverage for each scenario.**

**Source Code:**

Prints (int a, int b) { ------------ Printsum is a function

int result = a+ b;

If (result> 0)

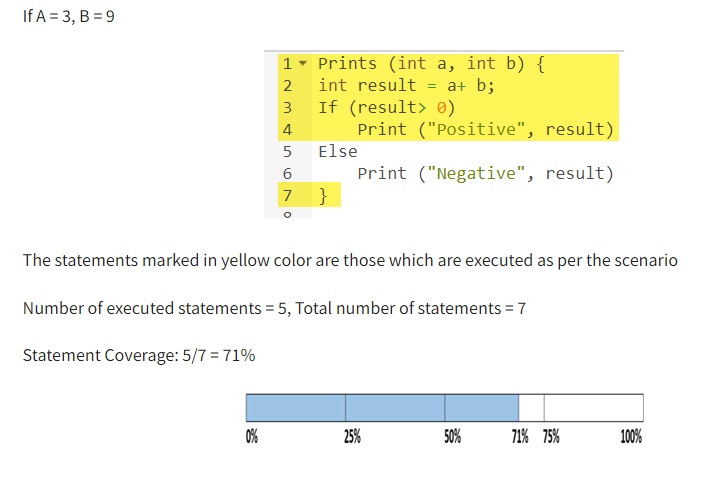
Print ("Positive", result)

Else

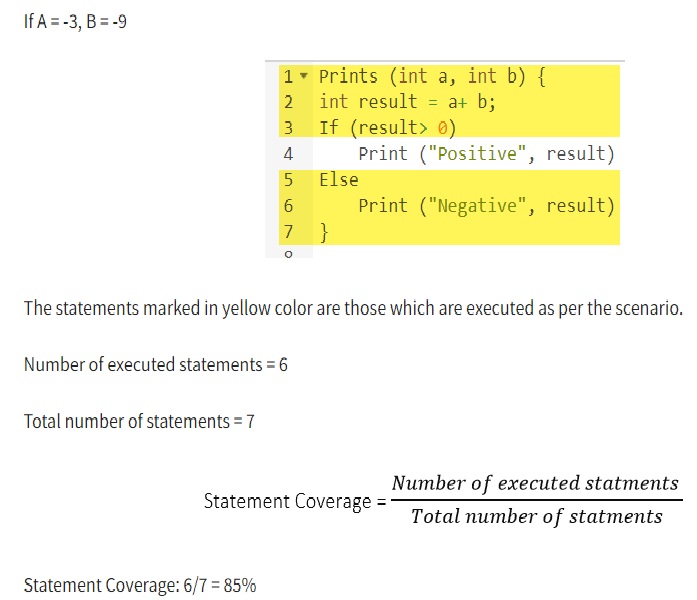
Print ("Negative", result)

} ----------- End of the source code

**Scenario 1**

****

**Scenario 2**

****

**But overall if you see, all the statements are being covered by 2nd scenario’s considered. So we can conclude that overall statement coverage is 100%.**

https://www.guru99.com/images/jsp/030116_0814_LearnStatem4.png

**What is covered by Statement Coverage?**

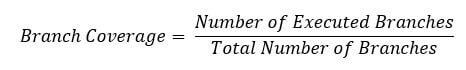
* 1. **Unused Statements**
  2. **Dead Code**
  3. **Unused Branches**
  4. **Missing Statements**
* **4.2.2. Branch & Decision coverage:-**

## Branch Coverage

**Branch Coverage** is a white box testing method in which every outcome from a code module(statement or loop) is tested. The purpose of branch coverage is to ensure that each decision condition from every branch is executed at least once. It helps to measure fractions of independent code segments and to find out sections having no branches.

For example, if the outcomes are binary, you need to test both True and False outcomes.

The formula to calculate Branch Coverage:



### Example of Branch Coverage

To learn branch coverage, let’s consider the same example used earlier

Consider the following code

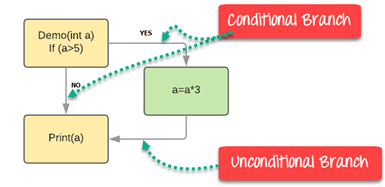
Demo(int a) {

If (a> 5)

a=a\*3

Print (a)

}



Branch Coverage will consider unconditional branch as well

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

**Advantages of Branch coverage:**

Branch coverage Testing offers the following advantages:

* Allows you to validate-all the branches in the code
* Helps you to ensure that no branched lead to any abnormality of the program’s operation
* Branch coverage method removes issues which happen because of statement coverage testing
* Allows you to find those areas which are not tested by other testing methods
* It allows you to find a quantitative measure of code coverage
* Branch coverage ignores branches inside the Boolean expressions
* **Decision Coverage :-**

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.

Decision coverage covers all possible outcomes of each and every Boolean condition of the code by using control flow graph or chart.

Generally, a decision point has two decision values one is true, and another is false that's why most of the times the total number of outcomes is two. The percent of decision coverage can be found by dividing the number of exercised outcome with the total number of outcomes and multiplied by 100.

Decision Coverage technique in whitebox testing link

In this technique, it is tough to get 100% coverage because sometimes expressions get complicated. Due to this, there are several different methods to report decision coverage. All these methods cover the most important combinations and very much similar to decision coverage. The benefit of these methods is enhancement of the sensitivity of control flow.

Decision Coverage is a white box testing technique which reports the true or false outcomes of each Boolean expression of the source code. The goal of decision coverage testing is to cover and validate all the accessible source code by checking and ensuring that each branch of every possible decision point is executed at least once.

In this coverage, expressions can sometimes get complicated. Therefore, it is very hard to achieve 100% coverage. That’s why there are many different methods of reporting this metric. All these methods focus on covering the most important combinations. It is very much similar to decision coverage, but it offers better sensitivity to control flow.

https://www.guru99.com/images/1/102518_1122_CodeCoverag12.jpg

* We can find the number of decision coverage as follows:-

Let's understand it by an example.

Consider the code to apply on decision coverage technique:

1. Test (int a)
2. {
3. If(a>4)
4. a=a\*3
5. Print (a)
6. }

**Scenario 1:**

**Value of a is 7 (a=7)**

1. Test (int a=7)
2. { if (a>4)
3. a=a\*3
4. print (a)
5. }

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.

Decision Coverage technique in whitebox testing link

Calculation of Decision Coverage percent:

Decision Coverage technique in whitebox testing link

1. Decision Coverage = ½\*100  (Only "True" is exercised)
2. =100/2
3. = 50
4. Decision Coverage is 50%

**Scenario 2:**

**Value of a is 3 (a=3)**

1. Test (int a=3)
2. { if (a>4)
3. a=a\*3
4. print (a)
5. }

The code highlighted in yellow will be executed. The outcome of this code is ?False? if condition (a>4) is checked.

Control flow graph when the value of a is 3

Decision Coverage technique in whitebox testing link

Calculation of Decision Coverage percent:

Decision Coverage technique in whitebox testing link

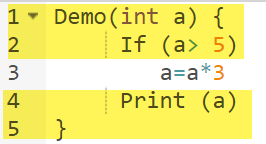
1. = ½\*100  (Only "False" is exercised) <br>
2. =100/2
3. = 50
4. Decision Coverage = 50%

**Result table of Decision Coverage:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case** | **Value of A** | **Output** | **Decision Coverage** |
| 1 | 3 | 3 | 50% |
| 2 | 7 | 21 | 50% |

**Scenario 3**

**Value of a is 2**

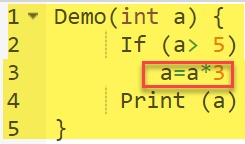


The code highlighted in yellow will be executed. Here the “No” outcome of the decision If (a>5) is checked.

Decision Coverage = 50%

**Scenario 4:**

**Value of a is 6**



The code highlighted in yellow will be executed. Here the “Yes” outcome of the decision If (a>5) is checked.

Decision Coverage = 50%

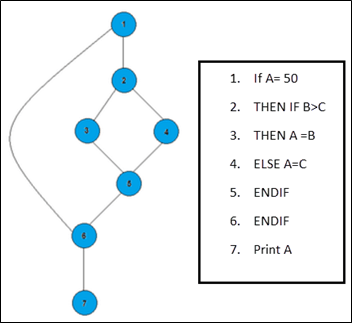
|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case** | **Value of A** | **Output** | **Decision Coverage** |
| 1 | 2 | 2 | 50% |
| 2 | 6 | 18 | 50% |

* **4.2.3 Path Coverage :-**
* In software program to reach from one start point to destination point there may be many paths. So, check each & every path, whether working properly or not.
* For this flow charts and data sheets are used.
* A white box test case design technique.
* Enables to derive the logical complexity measure of a procedural design, which can be used to define a basis set of execution paths.
* It gives all possible paths for which test case needs to be defined.
* This technique guarantees that every statement in a program is executed at least once during testing.
* Designs flow graph & measures the complexity by calculating the cyclomatric complexity of process.
* In flow graph notation each circle represents a Node.
* Node is representing the procedural statement.
* Flow graph notation terminology uses nodes, arrows, regions & Predicate nodes.
* Node represent process, arrow represent linksor edges, & Region is the area bounded by edges & nodes.
* Each node that contains a condition I.e. logical OR, AND etc is called a Predicate Node.
* Predicate nodes always have one incoming edge & two outgoing edges

Path testing is a structural testing method that involves using the source code of a program in order to find every possible executable path. It helps to determine all faults lying within a piece of code. This method is designed to execute all or selected path through a computer program.

Any software program includes, multiple entry and exit points. Testing each of these points is a challenging as well as time-consuming. In order to reduce the redundant tests and to achieve maximum test coverage, basis path testing is used.

Here we will take a simple example, to get a better idea what is basis path testing include



In the above example, we can see there are few conditional statements that is executed depending on what condition it suffice. Here there are 3 paths or condition that need to be tested to get the output,

Path 1: 1,2,3,5,6, 7

Path 2: 1,2,4,5,6, 7

Path 3: 1, 6, 7

## Steps for Basis Path testing

The basic steps involved in basis path testing include

* Draw a control graph (to determine different program paths)
* Calculate [Cyclomatic complexity](https://www.guru99.com/cyclomatic-complexity.html) (metrics to determine the number of independent paths)
* Find a basis set of paths
* Generate test cases to exercise each path

## Advantages of Basic Path Testing

* It helps to reduce the redundant tests
* It focuses attention on program logic
* It helps facilitates analytical versus arbitrary case design
* Test cases which exercise basis set will execute every statement in a program at least once
* **4.2.4. McCabe’s Cyclomatic Complexity Metric :-**

It is a quantitative measure of logical complexity , Defines independent paths in basis set of program Provides upper bound for number of tests conducted to ensure that all statements are executed at least once.

It is computed in one of three ways :

1.Cyclomatric complexity V(G) = Number of regions R

2.Cyclomatric complexity V(G) = No. of Predicate nodes P + 1

3.Cyclomatric complexity V(G) = E –N + 2

E = Number of Edges

N = Number of Nodes

* Cyclometric Complexity provides the quantitative measure of the logical complexity of a program.
* In basis path testing context the complexity value defines the number of independent paths in a program.
* This provides an upper bound for the number of tests that must be conducted to assure that all statements have been executed at least once.
* An independent path is any path through the program that introduces at least new set of processing statements or a new condition.
* In a flow graph, an independent path must move along at least one such edge which has not been traversed before the path is defined.

*Cyclomatic complexity is a software metric used to measure the complexity of a program. Thomas J. McCabe developed this metric in 1976.McCabe interprets a computer program as a set of a strongly connected directed graph*. Nodes represent parts of the source code having no branches and arcs represent possible control flow transfers during program execution. The notion of program graph has been used for this measure, and it is used to measure and control the number of paths through a program. The complexity of a computer program can be correlated with the topological complexity of a graph.

## How to Calculate Cyclomatic Complexity?

McCabe proposed the cyclomatic number, V (G) of graph theory as an indicator of software complexity. The cyclomatic number is equal to the number of linearly independent paths through a program in its graphs representation. For a program control graph G, cyclomatic number, V (G), is given as:

          V (G) = E - N + 2 \* P

E = The number of edges in graphs G

N = The number of nodes in graphs G

P = The number of connected components in graph G.

**Properties of Cyclomatic complexity:**

**Following are the properties of Cyclomatic complexity:**

1. V (G) is the maximum number of independent paths in the graph
2. V (G) >=1
3. G will have one path if V (G) = 1
4. Minimize complexity to 10

***Example :- Let a section of code as such:***

A = 10

IF B > C THEN

A = B

ELSE

A = C

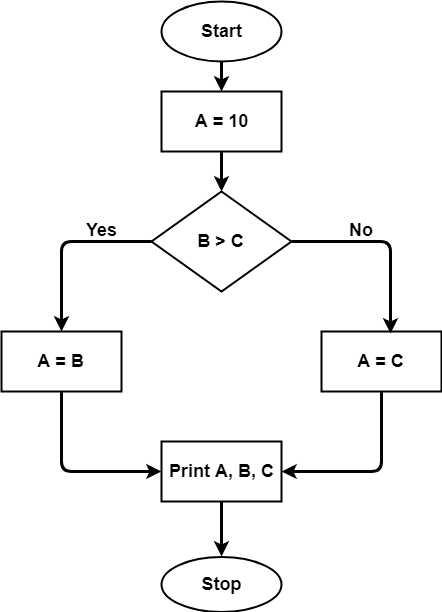
ENDIF

Print A

Print B

Print C

***Control Flow Graph of above code***



The cyclomatic complexity calculated for above code will be from control flow graph. The graph shows seven shapes(nodes), seven lines(edges), hence cyclomatic complexity is 7-7+2 = 2.

**Use of Cyclomatic Complexity:** 

Determining the independent path executions thus proven to be very helpful for Developers and Testers.

It can make sure that every path have been tested at least once.

Thus help to focus more on uncovered paths.

Code coverage can be improved.

Risk associated with program can be evaluated.

These metrics being used earlier in the program helps in reducing the risks.

* **Advantages of Cyclomatic Complexity:**.

It can be used as a quality metric, gives relative complexity of various designs.

It is able to compute faster than the Halstead’s metrics.

It is used to measure the minimum effort and best areas of concentration for testing.

It is able to guide the testing process.

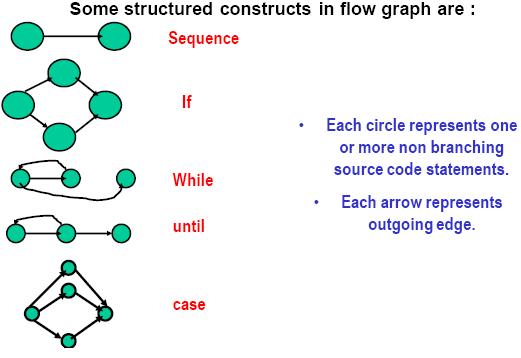
It is easy to apply.

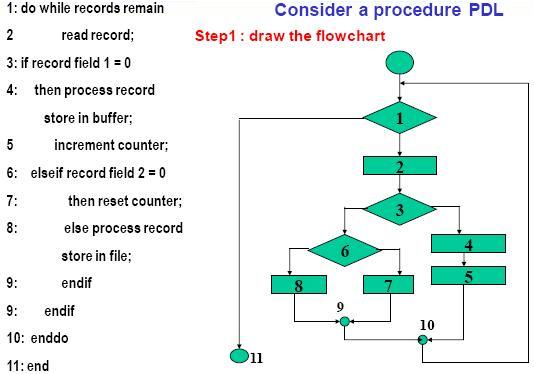
* **Disadvantages of Cyclomatic Complexity:-**

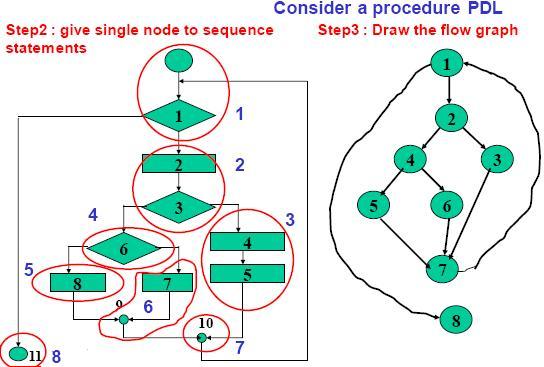
It is the measure of the programs’s control complexity and not the data the data complexity.

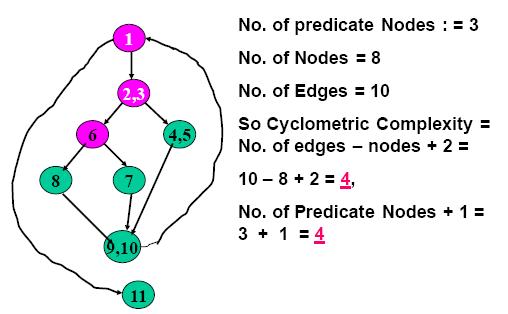
In this, nested conditional structures are harder to understand than non-nested structures.

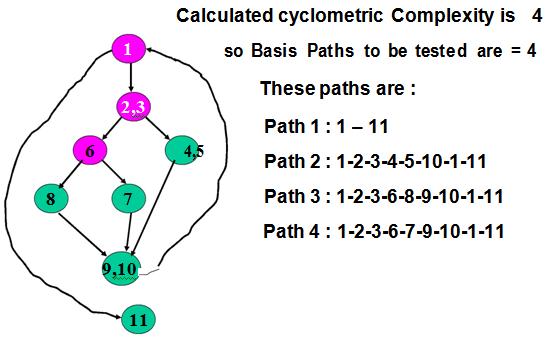
In case of simple comparisons and decision structures, it may give a misleading figure.





****

****

****

* **4.2.5 Data flow Based Testing :-**

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.

Data flow testing is a group of testing strategies to examine the control flow of programs in order to explore the sequence of variables according to the sequence of events. It mainly focuses on the points at which values assigned to the variables and the point at which these values are used by concentrating on both points, data flow can be tested.

Data flow testing uses the control flow graph to detect illogical things that can interrupt the flow of data. Anomalies in the flow of data are detected at the time of associations between values and variables due to:

* If the variables are used without initialization.
* If the initialized variables are not used at least once.

**Data Flow Testing** is a type of structural testing. It is a method that is used to find the test paths of a program according to the locations of definitions and uses of variables in the program. It has nothing to do with data flow diagrams.  
It is concerned with:

* Statements where variables receive values,
* Statements where these values are used or referenced.

To illustrate the approach of data flow testing, assume that each statement in the program assigned a unique statement number. For a statement number S-

DEF(S) = {X | statement S contains the definition of X}

USE(S) = {X | statement S contains the use of X}

If a statement is a loop or if condition then its DEF set is empty and USE set is based on the condition of statement s.

Data Flow Testing uses the control flow graph to find the situations that can interrupt the flow of the program.

Reference or define anomalies in the flow of the data are detected at the time of associations between values and variables. These anomalies are:

* A variable is defined but not used or referenced,
* A variable is used but never defined,
* A variable is defined twice before it is used

**Advantages of Data Flow Testing:**

Data Flow Testing is used to find the following issues-

* To find a variable that is used but never defined,
* To find a variable that is defined but never used,
* To find a variable that is defined multiple times before it is use,
* Deallocating a variable before it is used.

**Disadvantages of Data Flow Testing:-**

* Time consuming and costly process
* Requires knowledge of programming languages

**Example:**

**1.** read x, y;

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

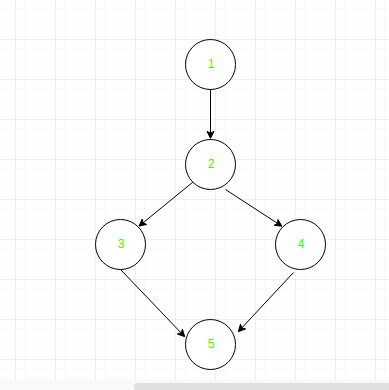
**3.** a = x+1

else

**4.** a = y-1

**5.** print a;

**Control flow graph of above example:**

****

* **4.2.6 Mutation Testing :-**

Mutation testing is a white box method in software testing where we insert errors purposely into a program (under test) to verify whether the existing test case can detect the error or not. In this testing, the mutant of the program is created by making some modifications to the original program.

The primary objective of mutation testing is to check whether each mutant created an output, which means that it is different from the output of the original program. We will make slight modifications in the mutant program because if we change it on a massive scale than it will affect the overall plan.

When we detected the number of errors, it implies that either the program is correct or the test case is inefficient to identify the fault.

Mutation testing purposes is to evaluate the quality of the case that should be able to fail the mutant code hence this method is also known as Fault-based testing as it used to produce an error in the program and that why we can say that the mutation testing is performed to check the efficiency of the test cases.

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

The changes made in the mutant program should be kept extremely small that it does not affect the overall objective of the program. Mutation Testing is also called Fault-based testing strategy as it involves creating a fault in the program and it is a type of [White Box Testing](https://www.guru99.com/white-box-testing.html) which is mainly used for [Unit Testing](https://www.guru99.com/unit-testing-guide.html).

Mutation was originally proposed in 1971 but lost due to the high costs involved. Now, again it has picked steam and is widely used for languages such as[Java](https://www.guru99.com/java-tutorial.html)and XML.

**What is mutation?**

The mutation is a small modification in a program; these minor modifications are planned to typical low-level errors which are happened at the time of coding process.

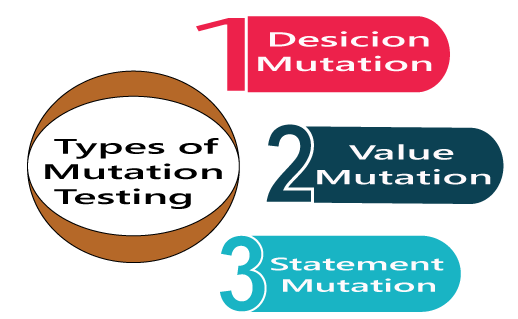
Generally, we deliberate the mutation operators in the form of rules which match the data and also generate some efficient environment to produce the mutant.

**Types of mutation testing**

Mutation testing can be classified into three parts, which are as follows:

* Decision mutations
* value mutations
* Statement mutations

Let us understand them one by one:



### Decision mutations

In this type of mutation testing, we will check the design errors. And here, we will do the modification in arithmetic and logical operator to detect the errors in the program.

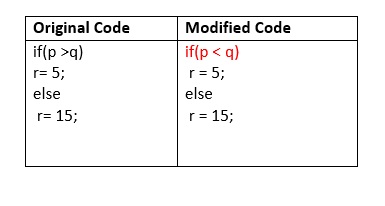
Like if we do the following changes in arithmetic operators:

* plus(+)→ minus(-)
* asterisk(\*)→ double asterisk(\*\*)
* plus(+)→incremental operator(i++)

Like if we do the following changes in logical operators

* Exchange P **>** → P**<**, OR P**>=**

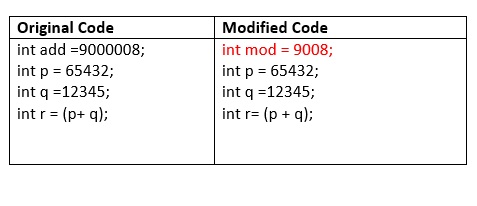
Now, let see one example for our better understanding:

****

### Value mutations

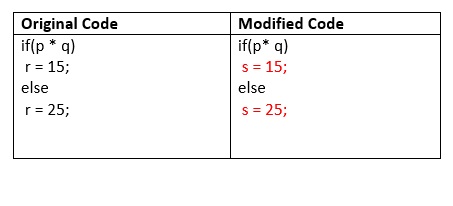
In this, the values will modify to identify the errors in the program, and generally, we will change the following:

* Small value à higher value
* Higher value àSmall value.

****

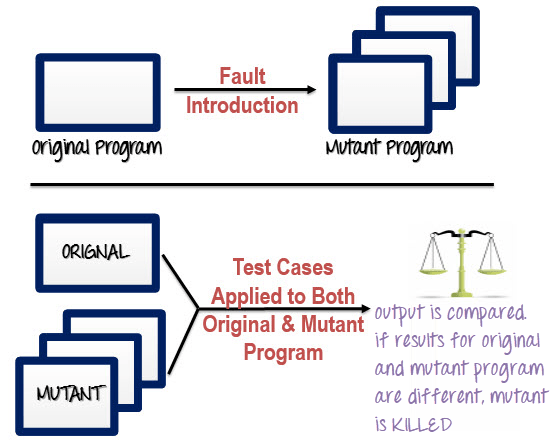
### Statement Mutations

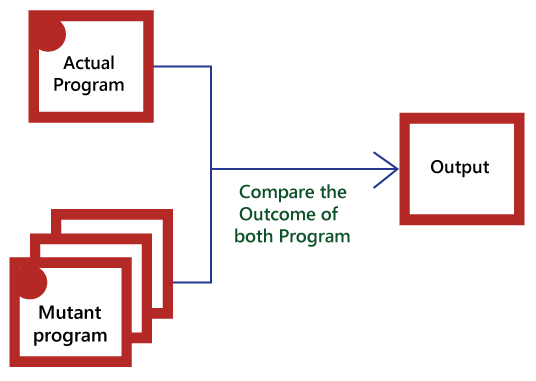
Statement mutations means that we can do the modifications into the statements by removing or replacing the line as we see in the below example:

****

In the above case, we have replaced the statement r=15 by s=15, and r=25 by s=25.

### How to execute Mutation Testing?

****



* In this, firstly, we will add the errors into the source code of the program by producing various versions, which are known mutants. Here every mutant having the one error, which leads the mutant kinds unsuccessful and also validates the efficiency of the test cases.
* After that, we will take the help of the test cases in the mutant program and the actual application will find the errors in the code.
* Once we identify the faults, we will match the output of the actual code and mutant code.
* After comparing the output of both actual and mutant programs, if the results are not matched, then the mutant is executed by the test cases. Therefore the test case has to be sufficient for identifying the modification between the actual program and the mutant program.
* And if the actual program and the mutant program produced the exact result, then the mutant is saved. And those cases are more active test cases because it helps us to execute all the mutants.

## Advantages and disadvantages of Mutation Testing

## Advantages:-

The benefits of mutation testing are as follows:

* It is a right approach for error detection to the application programmer
* The mutation testing is an excellent method to achieve the extensive coverage of the source program.
* Mutation testing helps us to give the most established and dependable structure for the clients.
* This technique can identify all the errors in the program and also helps us to discover the doubts in the code.

### Disadvantages:-

The drawbacks of mutant testing are as follows:

This testing is a bit of time taking and costlier process because we have many mutant programs that need to be created.

The mutation testing is not appropriate for Black-box testing as it includes the modification in the source code.

Every mutation will have the same number of test cases as compare to the actual program. Therefore the significant number of the mutant program may need to be tested beside the real test suite.

As it is a tedious process, so we can say that this testing requires the automation tools to test the application.

* **4.3. Test Design Techniques -Experience based techniques:-**
* **4.3.1. Error Guessing :-**

Error guessing is a technique in which there is no specific method for identifying the error. It is based on the experience of the test analyst, where the tester uses the experience to guess the problematic areas of the software. It is a type of black box testing technique which does not have any defined structure to find the error

As here possible bugs and defects are guessed in the areas where formal testing would not work. That’s why it is also called as experience based testing which has no specific method of testing.

In this approach, every test engineer will derive the values or inputs based on their understanding or assumption of the requirements, and we do not follow any kind of rules to perform error guessing technique.

The accomplishment of the error guessing technique is dependent on the ability and product knowledge of the tester because a good test engineer knows where the bugs are most likely to be, which helps to save lots of time.

Error guessing is a testing technique that makes use of a tester's skill, intuition and experience in testing similar applications to identify defects that may not be easy to capture by the more formal techniques. It is usually done after more formal techniques are completed.

**How does the error guessing technique be implemented?**

The implementation of this technique depends on the experience of the tester or analyst having prior experience with similar applications. It requires only well-experienced testers with quick error guessing technique. This technique is used to find errors that may not be easily captured by formal black box testing techniques, and that is the reason, it is done after all formal techniques.

The scope of the error guessing technique entirely depends on the tester and type of experience in the previous testing involvements because it does not follow any method and guidelines. Test cases are prepared by the analyst to identify conditions. The conditions are prepared by identifying most error probable areas and then test cases are designed for them.

The main purpose of this technique is to identify common errors at any level of testing by exercising the following tasks:

* Enter blank space into the text fields.
* Null pointer exception.
* Enter invalid parameters.
* Divide by zero.
* Use maximum limit of files to be uploaded.
* Check buttons without entering values.

The increment of test cases depends upon the ability and experience of the tester.

**Purpose of Error guessing**

The main purpose of the error guessing technique is to deal with all possible errors which cannot be identified as informal testing.

* The main purpose of error guessing technique is to deal with all possible errors which cannot be identified informal testing.
* It must contain the all-inclusive sets of test cases without skipping any problematic areas and without involving redundant test cases.
* This technique accomplishes the characteristics left incomplete during the formal testing.

Depending on the tester's intuition and experience, all the defects cannot be corrected. There are some factors that can be used by the examiner while using their experience -

* Tester's intuition
* Historical learning
* Review checklist
* Risk reports of the software
* Application UI
* General testing rules
* Previous test results
* Defects occurred in the past
* Variety of data which is used for testing
* Knowledge of AUT

## Examples of Error guessing method

### Example1

A function of the application requires a mobile number which must be of 10 characters. Now, below are the techniques that can be applied to guess error in the mobile number field:

* What will be the result, if the entered character is other than a number?
* What will be the result, if entered characters are less than 10 digits?
* What will be the result, if the mobile field is left blank?

After implementing these techniques, if the output is similar to the expected result, the function is considered to be bug-free, but if the output is not similar to the expected result, so it is sent to the development team to fix the defects.

However, error guessing is the key technique among all testing techniques as it depends on the experience of a tester, but it does not give surety of highest quality benchmark. It does not provide full coverage to the software. This technique can yield a better result if combined with other techniques of testing.

### Example2

Suppose we have one bank account, and we have to deposit some money over there, but the amount will be accepted on a particular range of **which is 5000-7000**. So here, we will provide the different input's value until it covers the maximum test coverage based on the error guessing technique, and see whether it is accepted or give the error message:

|  |  |
| --- | --- |
| **value** | **description** |
| 6000 | Accept |
| 5555 | Accept |
| 4000 | Error message |
| 8000 | Error message |
| blank | Error message |
| 100$ | Error message |
| ---- | ---- |
| ---- | ---- |
| Maximum test coverage |  |

**Condition: if amount >5000 and amount<7000 amount**

And, if we enter 5000 → error message (not accepted based on the condition)

7000→ error message (not accepted based on the condition)

## Advantages and disadvantage of Error guessing technique

### Advantages:-

*The benefits of error guessing technique are as follows*:

* It is a good approach to find the challenging parts of the software.
* It is beneficial when we will use this technique with the grouping of other formal testing techniques.
* It is used to enhance the formal test design techniques.
* With the help of this technique, we can disclose those bugs which were probably identified over extensive testing; therefore, the test engineer can save lots of time and effort.

### Disadvantage:-

*Following are the drawbacks of error guessing technique:*

The error guessing technique is person-oriented rather than process-oriented because it depends on the person's thinking.

If we use this technique, we may not achieve the minimum test coverage.

With the help of this, we may not cover all the input or boundary values.

With this, we cannot give the surety of the product quality.

The Error guessing technique can be done by those people who have product knowledge; it cannot be done by those who are new to the product.

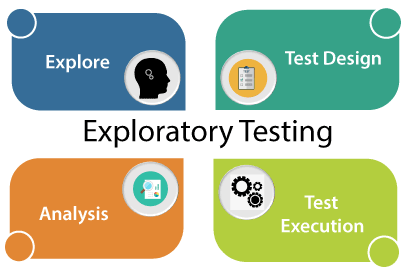
* **4.3.2 Exploratory Testing** :-

During testing phase where there is severe time pressure, Exploratory testing technique is adopted that combines the experience of testers along with a structured approach to testing.

Exploratory testing often performed as a black box testing technique, the tester learns things that together with experience and creativity generate new good tests to run.

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

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



## When we use exploratory testing:-

* When the requirement is missing
* Early iteration is required
* The testing team has the experienced testers when we have a critical application, and new testers entered into the team.

**For example,** to test any software or the application, first, we will perform unit, integration, and [system testing](https://www.javatpoint.com/system-testing).

So if we want to understand any application first, we will perform unit or component testing, suppose the application is having a login page having many elements, and we will understand each part and doing the component testing, but actually, we are doing the exploratory testing because we are exploring the application.

Suppose we have many modules in the application, and we are trying to do some integration scenarios.

Indirectly we are just doing exploratory testing while performing the [integration testing](https://www.javatpoint.com/integration-testing).

And, even if we are performing system testing, indirectly, we are performing exploratory testing because here we are also understanding and exploring the application.

## Why the requirement is missing

The requirement is missing because of the following reasons:

If the project is very old, the test engineer can't understand each scenario from the starting, and it might happen that the requirements will be missing.

**For example,** in each company, we don't see any fast process which means, we cannot expect the release to be done in just one month, and the product should be delivered in very less duration of time.

Many companies are still in the development phase for a particular product from the last 6 to 12 years.

Suppose one company has a 15-year-old project, and they hired a new test engineer now. The new test engineer faces many difficulties in understanding every scenario or requirement from scratch or starting because he/she is new with the application.

In that case, what test engineer will do with software that is 15 years old?

So firstly, he/she will take the application and start exploring the application. Once the test engineer starts using the application, he/she will get to know how the application is working. And, this process is nothing but exploratory testing.

## How to perform exploratory testing

To perform exploratory testing, first, we will start using the application and understand the requirement of the application from the person who has a good product knowledge such as senior test engineer, and developers.

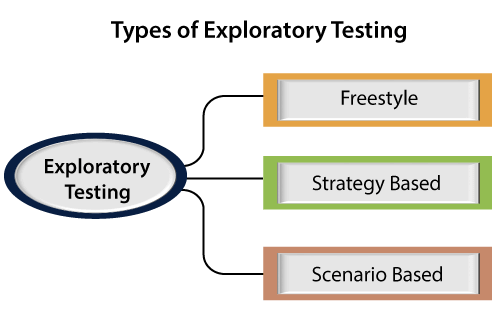
Then we will explore the application and write the necessary document, and this document is sent to the domain expert, and they will go through the document.

And we can test the application based on our knowledge, and taking the help of the competitive product, which is already launched in the market.

## Types of exploratory testing

Exploratory testing can be divided into three parts, which are as follows

* **Freestyle**
* **Strategy based**
* **Scenario-based**



### Freestyle :-

In freestyle testing, we did not follow any rules, there is no maximum coverage, and we will explore the application just like Adhoc testing.

If we want to get friendly with the software and checks the other test engineer's works, we can use freestyle exploratory testing.

### Strategy based :-

Strategy based exploratory testing can be performed with the help of multiple testing techniques such as risk-based, boundary value analysis, and equivalence partitioning.

It is done by the experienced tester and who is using the application for the longest time because he/she is known the application very much.

### Scenario-based:-

Scenario-based exploratory testing is performed with the help of multiple scenarios such as end-to-end, [test scenarios](https://www.javatpoint.com/test-scenario), and real user scenarios.

The test engineer can find defects and also checks various set of possibilities for the multiple scenarios with their application knowledge while they were exploring the application.

### Advantages and Disadvantages of Exploratory Testing :-

## Advantages:-

*Following are some benefits of exploratory testing:*

* If the test engineer using the exploratory testing, he/she may get a critical bug early because, in this testing, we need less preparation.
* In this testing, we can also find those bugs which may have been missed in the test cases.
* This testing can be used to test the new features, whereas, for the existing functionality, we will use the regression testing if we have less time to test the application.
* For the test engineer, this testing requires a lot of concentration to explore the application.

## Disadvantages:-

Following are the disadvantages of exploratory testing:

**Time Consuming**

It is a time taking process because we don't know the requirement, and which feature has to be tested first because we are just exploring the application.

The test engineer will misunderstand the feature as a bug.  
For example, suppose we have one login page and requirement says we have to provide the necessary details like username, password, and employee id then click on the login button.

But while performing exploratory testing, we only provide the details of username, password, and then click on the login button, but we have not entered the employee id. Since we don't have the requirement, and doing exploratory testing, that's why we feel that the employee id component is a bug, but it is a feature.

**Bugs can be misunderstood as a feature**

For example, suppose we have one registration page where we have to provide details like the username, password, mobile number, and the email **id**.

And the requirement says that when we are providing the mobile number and email id, the same code will be sent to the registered email id and mobile number to verify whether it is correct or not.

But when we are performing exploratory testing on the registration page and provide all the details (user name, password, mobile number, and email id), the code will only be sent to our mobile number, not to the email id.  
It is happening because the requirement is missing, and we will be misunderstood that this bug is a feature, and we never come to that this is a bug