**What is Statement coverage in testing?**

Statement coverage is a white box testing technique, which involves the execution of all the statements at least once in the source code. It is a metric, which is used to calculate and measure the number of statements in the source code which have been executed. Using this technique we can check what the source code is expected to do and what it should not. It can also be used to check the quality of the code and the flow of different paths in the program. The main drawback of this technique is that we cannot test the false condition in it.

**Example**:

1. Read A
2. Read B
3. if A>B
4. Print “A is greater than B”



1. else
2. Print "B is greater than A"
3. endif

Set1 :If A =5, B =2

No of statements Executed: 5

Total no of statements in the source code: 7

Statement coverage =5/7\*100 = 71.00 %

Set1 :If A =2, B =5

No of statements Executed: 6

Total no of statements in the source code: 7

Statement coverage =6/7\*100 = 85.20

(Statement coverage = No of statements Executed/Total no of statements in the source code \* 100)

This is purely a white box testing method. It tests the software’s internal coding and infrastructure and so the programmer is the one who should take the initiative to do this

Decision Coverage Testing

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

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

n 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.

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)

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

Decision Coverage = ½\*100  (Only "True" is exercised)

                    =100/2

                    = 50

Decision Coverage is 50%

**Scenario 2:**  
**Value of a is 3 (a=3**

Test (**int** a=3)

{ **if** (a>4)

a=a\*3

print (a)

}

. 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

= ½\*100  (Only "False" is exercised) <br>

=100/2

= 50

Decision Coverage = 50

## **Result table of Decision Coverage**

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

## **What is Condition Coverage Testing?**

Condition coverage is also known as Predicate Coverage in which each one of the Boolean expression have been evaluated to both TRUE and FALSE.

## **Example**

if ((A || B) && C)

{

print

}

else

{

Don’t print

}

## **Result**

In order to ensure complete Condition coverage criteria for the above example, A, B and C should be evaluated at least once against "true" an

So, in our example, the 3 following tests would be sufficient for 100% Condition coverage testing.

A = true | B = false | C = false

A = false | B = true | C = true

A = false | B = false | C = not eval

**Various examples in Basis Path Testing**

Difficulty Level : [Expert](https://www.geeksforgeeks.org/expert/)

Last Updated : 10 Jul, 2020

Prerequisite – [Basis Path Testing](https://www.geeksforgeeks.org/various-path-testing-in-software-engineering/" \t "https://www.geeksforgeeks.org/example-basis-path-testing-white-box-testing/_blank)  
We have seen the steps involved in designing the test cases for a program using the basis path testing in the previous [article](https://www.geeksforgeeks.org/various-path-testing-in-software-engineering/" \t "https://www.geeksforgeeks.org/example-basis-path-testing-white-box-testing/_blank). Now, let’s solve an example following the same steps.

Question : Consider the given program that checks if a number is prime or not. For the following program :

Draw the Control Flow Graph

Calculate the Cyclomatic complexity using all the methods

List all the Independent Paths

Design test cases from independent paths

|  |
| --- |
| int main()  {      int n, index;      cout << "Enter a number: " << endl;      cin >> n;      index = 2;      while (index <= n - 1) {          if (n % index == 0) {              cout << "It is not a prime number" << endl;              break;          }          index++;      }      if (index == n)          cout << "It is a prime number" << endl;  } // end main |

Solution :  
1. Draw the Control Flow Graph –

Step-1:  
Start numbering the statements after declaration of the variables (if no variables have been initialized in that statement). However, if a variable has been initialized and declared in the same line, then numbering should start from that line itself.

For the given program, this is how numbering will be done:

int main()

{

int n, index;

1 cout << "Enter a number: " <> n;

3 index = 2;

4 while (index <= n - 1)

5 {

6 if (n % index == 0)

7 {

8 cout << "It is not a prime number" << endl;

9 break;

10 }

11 index++;

12 }

13 if (index == n)

14 cout << "It is a prime number" << endl;

15 } // end main

Step-2:  
Put the sequential statements into one single node. For example, statements 1, 2 and 3 are all sequential statements and hence should be combined into a single node. And for other statements, we will follow the notations as discussed [here](https://www.geeksforgeeks.org/various-path-testing-in-software-engineering/" \t "https://www.geeksforgeeks.org/example-basis-path-testing-white-box-testing/_blank).

Note –  
Use alphabetical numbering on nodes for simplicity.

The graph obtained will be as follows :

