**What is Code coverage?**

Code coverage is a measure which describes the degree of which the source code of the program has been tested. It is one form of white box testing which finds the areas of the program not exercised by a set of test cases. It also creates some test cases to increase coverage and determining a quantitative measure of code coverage.

In most cases, code coverage system gathers information about the running program. It also combines that with source code information to generate a report about the test suite's code coverage.

## Why use Code Coverage?

Here, are some prime reasons for using code coverage:

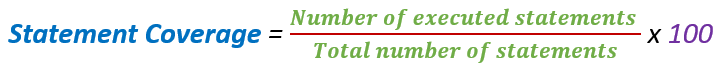
* It helps you to measure the efficiency of test implementation
* It offers a quantitative measurement.
* It defines the degree to which the source code has been tested.

**Statement Coverage**

What is Statement Coverage?

Statement coverage is a white box test design technique which involves execution of all the executable statements in the source code at least once. It is used to calculate and measure the number of statements in the source code which can be executed given the requirements.

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

[](https://www.guru99.com/images/jsp/030116_0814_LearnStatem1.png)

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)

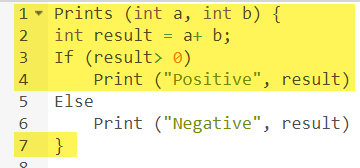
Else

Print ("Negative", result)

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

**Scenario 1:**

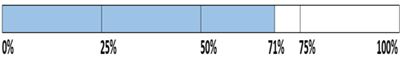
If A = 3, B = 9

[](https://www.guru99.com/images/1/102518_1122_CodeCoverag2.png)

The statements marked in yellow color are those which are executed as per the scenario

Number of executed statements = 5, Total number of statements = 7

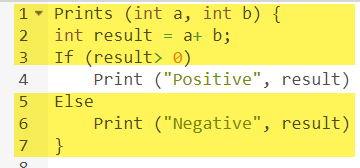
Statement Coverage: 5/7 = 71%

[](https://www.guru99.com/images/jsp/030116_0814_LearnStatem2.png)

Likewise we will see scenario 2,

**Scenario 2:**

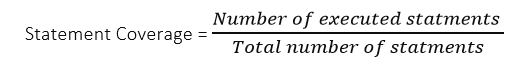
If A = -3, B = -9

[](https://www.guru99.com/images/1/102518_1122_CodeCoverag4.png)

The statements marked in yellow color are those which are executed as per the scenario.

Number of executed statements = 6

Total number of statements = 7

[](https://www.guru99.com/images/jsp/030116_0814_LearnStatem6.png)

Statement Coverage: 6/7 = 85%

[https://www.guru99.com/images/jsp/030116_0814_LearnStatem3.png](https://www.guru99.com/images/jsp/030116_0814_LearnStatem3.png)

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](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

## Decision Coverage

Decision coverage reports the true or false outcomes of each Boolean expression. 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](https://www.guru99.com/images/1/102518_1122_CodeCoverag12.jpg)

### Example of decision coverage

Consider the following code-

Demo(int a) {

If (a> 5)

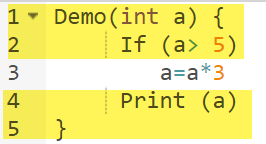
a=a\*3

Print (a)

}

**Scenario 1:**

Value of a is 2

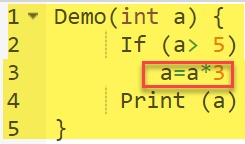
[](https://www.guru99.com/images/1/102518_1122_CodeCoverag8.png)

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

Decision Coverage = 50%

**Scenario 2:**

Value of a is 6

[](https://www.guru99.com/images/1/102518_1122_CodeCoverag9.png)

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% |

## Branch Coverage

In the branch coverage, every outcome from a code module is tested. For example, if the outcomes are binary, you need to test both True and False outcomes.

It helps you to ensure that every possible branch from each decision condition is executed at least a single time.

By using Branch coverage method, you can also measure the fraction of independent code segments. It also helps you to find out which is sections of code don't have any branches.

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

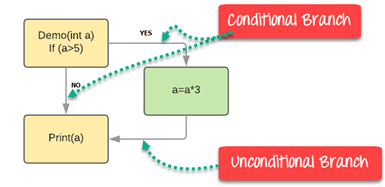
Demo(int a) {

If (a> 5)

a=a\*3

Print (a)

}

[](https://www.guru99.com/images/1/102518_1122_CodeCoverag10.png)

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

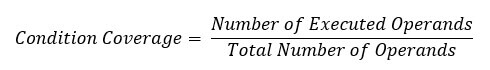
## Condition Coverage

Conditional coverage or expression coverage will reveal how the variables or subexpressions in the conditional statement are evaluated. In this coverage expressions with logical operands are only considered.

For example, if an expression has Boolean operations like AND, OR, XOR, which indicated total possibilities.

Conditional coverage offers better sensitivity to the control flow than decision coverage. Condition coverage does not give a guarantee about full decision coverage

The formula to calculate Condition Coverage:

[](https://www.guru99.com/images/1/102518_1122_CodeCoverag14.jpg)

Example:

[https://www.guru99.com/images/1/102518_1122_CodeCoverag11.png](https://www.guru99.com/images/1/102518_1122_CodeCoverag11.png)

For the above expression, we have 4 possible combinations

* TT
* FF
* TF
* FT

Consider the following input

|  |  |  |  |
| --- | --- | --- | --- |
| X=3  Y=4 | (x<y) | TRUE | Condition Coverage is ¼ = 25% |
| A=3  B=4 | (a>b) | FALSE |