Levels of Testing

## What is Unit Testing?

**UNIT TESTING** is a type of software testing where individual units or components of a software are tested. The purpose is to validate that each unit of the software code performs as expected. Unit Testing is done during the development (coding phase) of an application by the developers. Unit Tests isolate a section of code and verify its correctness. A unit may be an individual function, method, procedure, module, or object.

In SDLC, STLC, V Model, Unit testing is first level of testing done before integration testing. Unit testing is a WhiteBox testing technique that is usually performed by the developer. Though, in a practical world due to time crunch or reluctance of developers to tests, QA engineers also do unit testing.

## Why Unit Testing?

**Unit Testing** is important because software developers sometimes try saving time doing minimal unit testing and this is myth because inappropriate unit testing leads to high cost[Defect](https://www.guru99.com/defect-management-process.html)fixing during [System Testing](https://www.guru99.com/system-testing.html), [Integration Testing](https://www.guru99.com/integration-testing.html) and even Beta Testing after application is built. If proper unit testing is done in early development, then it saves time and money in the end.

**How to do Unit Testing**

In order **to do Unit Testing**, developers write a section of code to test a specific function in software application. Developers can also isolate this function to test more rigorously which reveals unnecessary dependencies between function being tested and other units so the dependencies can be eliminated. Developers generally use [UnitTest framework](https://www.guru99.com/test-automation-framework.html) to develop automated test cases for unit testing.

Unit Testing is of two types

* Manual
* Automated

Unit testing is commonly automated but may still be performed manually. Software Engineering does not favor one over the other but automation is preferred. A manual approach to unit testing may employ a step-by-step instructional document.

Under the automated approach-

* A developer writes a section of code in the application just to test the function. They would later comment out and finally remove the test code when the application is deployed.
* A developer could also isolate the function to test it more rigorously. This is a more thorough unit testing practice that involves copy and paste of code to its own testing environment than its natural environment. **Isolating the code helps in revealing unnecessary dependencies between the code being tested and other units or data spaces** in the product. These dependencies can then be eliminated.
* A coder generally uses a UnitTest Framework to develop automated test cases. Using an automation framework, the developer codes criteria into the test to verify the correctness of the code. During execution of the test cases, the framework logs failing test cases. Many frameworks will also automatically flag and report, in summary, these [failed test cases](https://www.guru99.com/run-failed-test-cases-in-testng.html). Depending on the severity of a failure, the framework may halt subsequent testing.
* The workflow of Unit Testing is 1) Create Test Cases 2) Review/Rework 3) Baseline 4) Execute Test Cases.

**Unit Testing Techniques**

The **Unit Testing Techniques** are mainly categorized into three parts which are Black box testing that involves testing of user interface along with input and output, White box testing that involves testing the functional behaviour of the software application and Gray box testing that is used to execute test suites, test methods, test cases and performing risk analysis.

Code coverage techniques used in Unit Testing are listed below:

* Statement Coverage
* Decision Coverage
* Branch Coverage
* Condition Coverage
* Finite State Machine Coverage

## 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.

## Decision Coverage

**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.

### Example of decision coverage

Consider the following code-

Demo(int a) {

If (a> 5)

a=a\*3

Print (a)

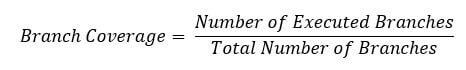
}

## 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

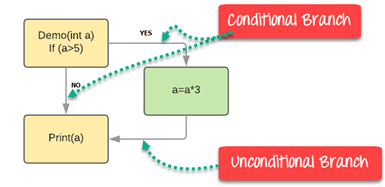
Demo(int a) {

If (a> 5)

a=a\*3

Print (a)

}



## Condition Coverage

**Condition Coverage** or expression coverage is a testing method used to test and evaluate the variables or sub-expressions in the conditional statement. The goal of condition coverage is to check individual outcomes for each logical condition. Condition coverage offers better sensitivity to the control flow than decision coverage. In this coverage, expressions with logical operands are only considered.

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

Consider the following input

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

Consider the following input

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