**What is Unit Testing?**

Unit Testing of software applications is done during the development (coding) of an application.

The objective of Unit Testing is to isolate a section of code and verify its correctness. In procedural programming a unit may be an individual function or procedure

The goal of Unit Testing is to isolate each part of the program and show that the individual parts are correct. Unit Testing is usually performed by the developer. 

**Why do Unit Testing? Why it is important?**

Sometimes software developers attempt to save time by doing minimal unit testing. This is a myth because skimping on unit testing leads to higher Defect fixing costs during System Testing, Integration Testing and even Beta Testing after the application is completed. Proper unit testing done during the development stage saves both time and money in the end

**How to Create Unit Test Cases**

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

Under the automated approach-

* A developer could write another 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 done.
* They could also isolate the function to test it more rigorously. This is a more thorough unit testing practice that involves copy and pasting the function to its own testing environment to other 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 may use 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 unit. During execution of the test cases, the framework logs those that fail any criterion. Many frameworks will also automatically flag and report in a summary these failed test cases. Depending upon the severity of a failure, the framework may halt subsequent testing.

**Mock Objects**

Unit testing relies on mock objects being created to test sections of code that are not yet part of a complete application. Mock objects fill in for the missing parts of the program. For example, you might have a function that needs variables or objects that are not created yet. In unit testing, those will be accounted for in the form of mock objects created solely for the purpose of the unit testing done on that section of code.

**Unit Testing Tools**

There are several automated tools available to assist with unit testing. We will provide a few examples below:

* Rational Software - Rational Software by IBM has a unittest feature known as "Rational Test Realtime". The software contains a complete range of testing tools for much more than just unit testing. It is used for Ada, Java, C and C++. It creates unit tests by reverse engineering the software. Operating systems it supports include Windows, Linux, Solaris, HP-UX and AIX. Go to http://www-01.ibm.com/software/rational/ to learn more.
* JavaScript Assertion Unit- Also known as jsAsserUnit, this Freeware JavaScript unit testing tool can be used on any platform that supports JavaScript. It is available at http://jsassertunit.sourceforge.net/docs/index.html
* CUT - CUT is a Freeware unittest tool for C, C++ and Objective C. It is great for embedded software testing frameworks and desktop applications on Linux and Windows operating systems. Learn more at sourceforge.net by going to http://sourceforge.net/projects/cut/.
* Dotunit - Dotunit is a .net framework Freeware unit testing tool. Part of Junit on the Microsoft .net framework, Dotunit is used for automating unit testing on windows systems. This is another tool from sourceforge.net, so look for it at: http://dotunit.sourceforge.net/

Those are just a few of the available unit testing tools. There are lots more, especially for C languages and Java, but you are sure to find a unit testing tool for your programming needs regardless of the language you use.

**Extreme Programming & Unit Testing**

Unit testing in Extreme Programming involves the extensive use of testing frameworks. A unit test framework is used in order to create automated unit tests. Unit testing frameworks are not unique to extreme programming, but they are essential to it. Below we look at some of what extreme programming brings to the world of unit testing:

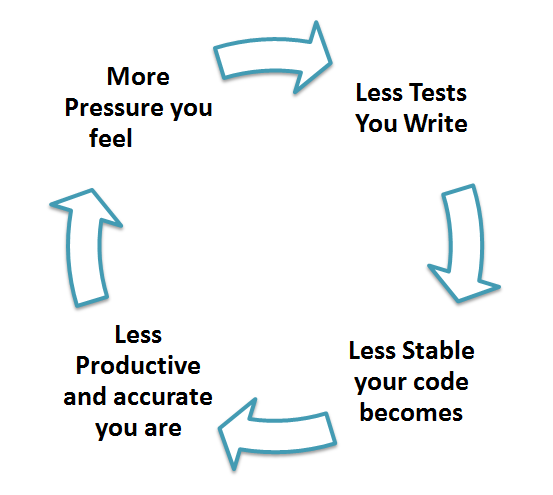
* Tests are written before the code
* Rely heavily on testing frameworks
* All classes in the applications are tested
* Quick and easy integration is made possible

**Unit Testing Myth**

Myth: It requires time and I am always overscheduled

My code is rock solid! I do not need unit tests.

Myths by their very nature are false assumptions. These assumptions lead to a vicious cycle as follows -

[](https://cdn.guru99.com/images/unit-testing.png)

Truth is Unit testing increase the speed of development.

Programmers think that Integration Testing will catch all errors and do not unit test. Once units are integrated, very simple errors which could have very easily found and fixed in unit tested take very long time to be traced and fixed.

**Unit Testing Benefits and Advantage**

* Developers looking to learn what functionality is provided by a unit and how to use it can look at the unit tests to gain a basic understanding of the unit API.
* Unit testing allows the programmer to refactor code at a later date, and make sure the module still works correctly (i.e. Regression testing). The procedure is to write test cases for all functions and methods so that whenever a change causes a fault, it can be quickly identified and fixed.
* Due to the modular nature of the unit testing, we can tests parts of project without waiting for others to be completed.

**Unit Testing Limitations**

* Unit testing can't be expected to catch every error in a program. It is not possible to evaluate all execution paths even in the most trivial programs
* Unit testing by its very nature focuses on a unit of code. Hence it can't catch integration errors or broad system level errors.

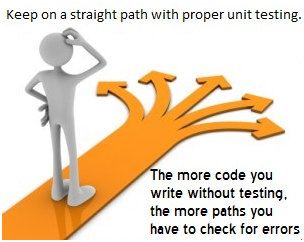
It's recommended unit testing be used in conjunction with other testing activities.

**Unit Testing Techniques**

* Structural Techniques
* Functional Testing Techniques
* Error Based Techniques

**Unit Testing Best Practices**

* Unit Test cases should be independent. In case of any enhancements or change in requirements, unit test cases should not be affected.
* Test only one code at a time.
* Follow clear and consistent naming conventions for your unit tests
* In case of change in code in any module, ensure there is a corresponding unit Test Case for the module and the module passes the tests before changing the implementation
* Bugs identified during unit testing must be fixed before proceeding to the next phase in SDLC
* Adopt a "test as your code" approach. The more code you write without testing the more paths you have to check for errors.

[](https://cdn.guru99.com/images/unit_testing_best_practise.png)

**Summary**

As you can see, there can be a lot involved in unit testing. It can be complex or rather simple depending on the application being tested and the testing strategies, tools and philosophies used. Unit testing is always necessary on some level. That is a certainty.