# Implementation and Testing

## code

## Testing Approach

**White Box Testing**

**WHITE BOX TESTING** (also known as Clear Box Testing, Open Box Testing, Glass Box Testing, Transparent Box Testing, Code-Based Testing or Structural Testing) is a [software](http://softwaretestingfundamentals.com/software-testing-methods/) [testing method](http://softwaretestingfundamentals.com/software-testing-methods/) in which the internal structure/design/implementation of the item being tested is known to the tester. The tester chooses inputs to exercise paths through the code and determines the appropriate outputs. Programming know-how and the implementation knowledge is essential. White box testing is testing beyond the user interface and into the nitty-gritty of a system.

This method is named so because the software program, in the eyes of the tester, is like a white/transparent box; inside which one clearly sees.

Definition by ISTQB

* + - **white-box testing:** Testing based on an analysis of the internal structure of the component or system.
    - **white-box test design technique:** Procedure to derive and/or select test cases based on an analysis of the internal structure of a component or system.

Example:

A tester, usually a developer as well, studies the implementation code of a certain field on a webpage, determines all legal (valid and invalid) AND illegal inputs and verifies the outputs against the expected outcomes, which is also determined by studying the implementation code.

White Box Testing is like the work of a mechanic who examines the engine to see why the car is not moving.

Levels Applicable To

White Box Testing method is applicable to the following levels of software testing:

* + - [Unit Testing](http://softwaretestingfundamentals.com/unit-testing/): For testing paths within a unit.
    - [Integration Testing](http://softwaretestingfundamentals.com/integration-testing/): For testing paths between units.
    - [System Testing](http://softwaretestingfundamentals.com/system-testing/): For testing paths between subsystems.

However, it is mainly applied to Unit Testing.

Advantages

* + - Testing can be commenced at an earlier stage. One need not wait for the GUI to be available.
    - Testing is more thorough, with the possibility of covering most paths.

Disadvantages

* + - Since tests can be very complex, highly skilled resources are required, with a thorough knowledge of programming and implementation.
    - Test script maintenance can be a burden if the implementation changes too frequently.
    - Since this method of testing is closely tied to the application being tested, tools to cater to every kind of implementation/platform may not be readily available.

## Unit Testing

UNIT TESTING is a level of software testing where individual units/ components of a software are tested. The purpose is to validate that each unit of the software performs as designed. A unit is the smallest testable part of any software. It usually has one or a few inputs and usually a single output. In procedural programming, a unit may be an individual program, function, procedure, etc. In object-oriented programming, the smallest unit is a method, which may belong to a base/ super class, abstract class or derived/ child class. (Some treat a module of an application as a unit. This is to be discouraged as there will probably be many individual units within that module.) Unit testing frameworks, drivers, stubs, and mock/ fake objects are used to assist in unit testing.

Definition by ISTQB

* + - * **unit testing:** See component testing.
      * **component testing:** The testing of individual software components.

Unit Testing Method

It is performed by using the [White Box Testing](http://softwaretestingfundamentals.com/white-box-testing/) method.

When is it performed?

Unit Testing is the first [level of software testing](http://softwaretestingfundamentals.com/software-testing-levels/) and is performed prior to [Integration Testing](http://softwaretestingfundamentals.com/integration-testing/).

Who performs it?

It is normally performed by software developers themselves or their peers. In rare cases, it may also be performed by independent software testers.

Unit Testing Benefits

* + - * Unit testing increases confidence in changing/ maintaining code. If good unit tests are written and if they are run every time any code is changed, we will be able to promptly catch any defects introduced due to the change. Also, if codes are already made less interdependent to make unit testing possible, the unintended impact of changes to any code is less.
      * Codes are more reusable. In order to make unit testing possible, codes need to be modular. This means that codes are easier to reuse.
      * Development is faster. How? If you do not have unit testing in place, you write your code and perform that fuzzy ‘developer test’ (You set some breakpoints, fire up the GUI, provide a few inputs that hopefully hit your code and hope that you are all set.) But,

if you have unit testing in place, you write the test, write the code and run the test. Writing tests takes time but the time is compensated by the less amount of time it takes to run the tests; You need not fire up the GUI and provide all those inputs. And, of course, unit tests are more reliable than ‘developer tests’. Development is faster in the long run too. How? The effort required to find and fix defects found during unit testing is very less in comparison to the effort required to fix defects found during system testing or acceptance testing.

* + - * The cost of fixing a defect detected during unit testing is lesser in comparison to that of defects detected at higher levels. Compare the cost (time, effort, destruction, humiliation) of a defect detected during acceptance testing or when the software is live.
      * Debugging is easy. When a test fails, only the latest changes need to be debugged. With testing at higher levels, changes made over the span of several days/weeks/months need to be scanned.
      * Codes are more reliable. Why? I think there is no need to explain this to a sane person.

## Integration Testing

**INTEGRATION TESTING** is a level of software testing where individual units are combined and tested as a group. The purpose of this level of testing is to expose faults in the interaction between integrated units. Test drivers and test stubs are used to assist in Integration Testing.

Definition by ISTQB

* + - * **integration testing:** Testing performed to expose defects in the interfaces and in the interactions between integrated components or systems. See also component integration testing, system integration testing.
      * **component integration testing:** Testing performed to expose defects in the interfaces and

interaction between integrated components.

* + - * **system integration testing:** Testing the integration of systems and packages; testing interfaces to external organizations (e.g. Electronic Data Interchange, Internet)

When is it performed?

System Testing is the third [level of software testing](http://softwaretestingfundamentals.com/software-testing-levels/) performed after [Integration Testing](http://softwaretestingfundamentals.com/integration-testing/) and before [Acceptance Testing.](http://softwaretestingfundamentals.com/acceptance-testing/)

Who performs it?

Normally, independent Testers perform System Testing

## Test Cases , Test Data & Test Result

Test cases should be designed and written by someone who understands the function or technology being tested. A test case should include the following information –

1. Purpose of the test
2. Software requirements and Hardware requirements (if any)
3. Specific setup or configuration requirements
4. Description on how to perform the test(s)
5. Expected results or success criteria for the test

Designing test cases can be time consuming in a testing schedule, but they are worth giving time because they can really avoid unnecessary retesting or debugging or at least lower it.

Organizations can take the test cases approach in their own context and according to their own perspectives. Some follow a general step way approach while others may opt for a more detailed and complex approach.

It is very important for you to decide between the two extremes and judge on what would work the best for you.

Designing proper test cases is very vital for your software testing plans as a lot of bugs, ambiguities, inconsistencies and slip ups can be recovered in time as also it helps in saving your time on continuous debugging and re-testing test cases.