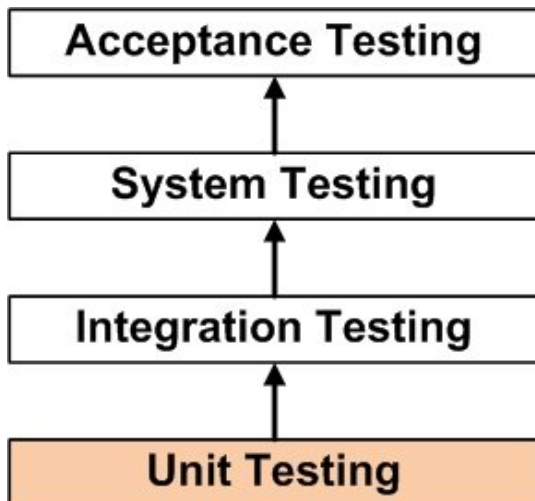




Unit Testing

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 Tasks

- Unit Test Plan
 - Prepare
 - Review
 - Rework
 - Baseline
- Unit Test Cases/Scripts
 - Prepare
 - Review
 - Rework
 - Baseline
- Unit Test
 - Perform

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.

Unit Testing Tips

- Find a tool/framework for your language.
- Do not create test cases for everything. Instead, focus on the tests that impact the behavior of the system.
- Isolate the development environment from the test environment.
- Use test data that is close to that of production.
- Before fixing a defect, write a test that exposes the defect. Why? First, you will later be able to catch the defect if you do not fix it properly. Second, your test suite is now more comprehensive. Third, you will most probably be too lazy to write the test after you have already fixed the defect.
- Write test cases that are independent of each other. For example, if a class depends on a database, do not write a case that interacts with the database to test the class. Instead, create an abstract interface around that database connection and implement that interface with a mock object.
- Aim at covering all paths through the unit. Pay particular attention to loop conditions.
- Make sure you are using a version control system to keep track of your test scripts.
- In addition to writing cases to verify the behavior, write cases to ensure the performance of the code.
- Perform unit tests continuously and frequently.

One more reason

Let's say you have a program comprising of two units and the only test you perform is system testing (<http://softwaretestingfundamentals.com/system-testing/>). [You skip unit and integration testing (<http://softwaretestingfundamentals.com/integration-testing/>).] During testing, you find a bug. Now, how will you determine the cause of the problem?

- Is the bug due to an error in unit 1?
- Is the bug due to an error in unit 2?
- Is the bug due to errors in both units?
- Is the bug due to an error in the interface between the units?
- Is the bug due to an error in the test or test case?

Unit testing is often neglected but it is, in fact, the most important level of testing.

◀ Integration Testing (<http://softwaretestingfundamentals.com/integration-testing/>)

Software Test Manager Sample Job Description ▶ (<http://softwaretestingfundamentals.com/software-test-manager-sample-job-description/>) ^

Starting to Test with TestRail

Get a first glance of TestRail with our live webinar and learn from the best.

TestRail

OPEN

Search...



Red Hat

**The future of
financial services
is open**

[Learn more](#)



Enter your email address:

[Subscribe](#)


If you do not find the verification email, check your spam folder


Delivered by FeedBurner (<https://feedburner.google.com>)





Categories


 [Artifacts \(http://softwaretestingfundamentals.com/category/artifacts/\)](http://softwaretestingfundamentals.com/category/artifacts/)

 [Basics \(http://softwaretestingfundamentals.com/category/basics/\)](http://softwaretestingfundamentals.com/category/basics/)


 [Career \(http://softwaretestingfundamentals.com/category/career/\)](http://softwaretestingfundamentals.com/category/career/)

 [Defects \(http://softwaretestingfundamentals.com/category/defects/\)](http://softwaretestingfundamentals.com/category/defects/)

 [Levels \(http://softwaretestingfundamentals.com/category/levels/\)](http://softwaretestingfundamentals.com/category/levels/)

 [Methods \(http://softwaretestingfundamentals.com/category/methods/\)](http://softwaretestingfundamentals.com/category/methods/)

 [Metrics \(http://softwaretestingfundamentals.com/category/metrics/\)](http://softwaretestingfundamentals.com/category/metrics/)

 [Resources \(http://softwaretestingfundamentals.com/category/resources/\)](http://softwaretestingfundamentals.com/category/resources/)

 [Types \(http://softwaretestingfundamentals.com/category/types/\)](http://softwaretestingfundamentals.com/category/types/)

Skim

[Disclaimer \(http://softwaretestingfundamentals.com/disclaimer/\)](http://softwaretestingfundamentals.com/disclaimer/)

[Privacy Policy \(http://softwaretestingfundamentals.com/privacy-policy/\)](http://softwaretestingfundamentals.com/privacy-policy/)

Engage

[Facebook Page \(https://www.facebook.com/softwareTestingFundamentals/\)](https://www.facebook.com/softwareTestingFundamentals/)

[Advertise \(http://softwaretestingfundamentals.com/advertise/\)](http://softwaretestingfundamentals.com/advertise/)

[Contact \(http://softwaretestingfundamentals.com/contact/\)](http://softwaretestingfundamentals.com/contact/)

Ponder

Weinberg's Second Law: If builders built buildings the way programmers wrote programs, then the first woodpecker that came along would have destroyed civilization.