Do you know how developers test their code? What methods do they use to test before releasing the code? The answer to all this is ***Unit testing***. Unit testing is also known as ***Module, Program***, or ***Component testing***. In this article, we will discuss the importance of Component testing so that development and testing teams can work more collaboratively to design, test, and launch a bug-free software.

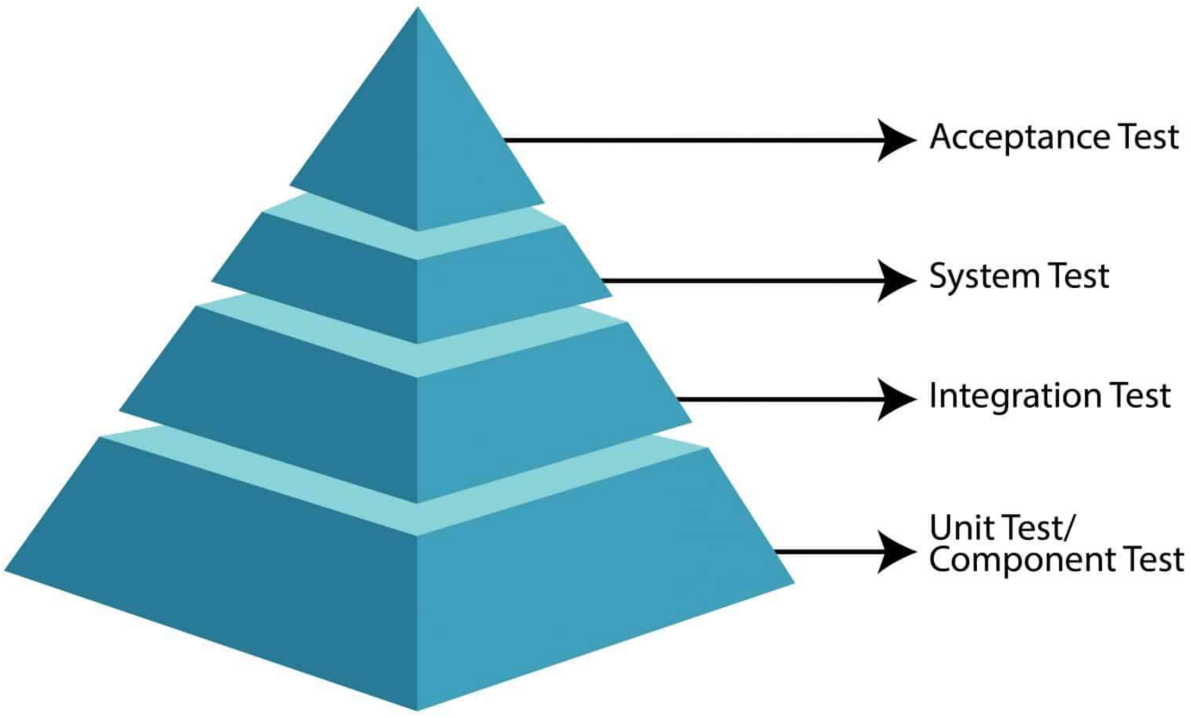
Introduction to Component Testing?
What are its Objectives?
What are the typical defects and failures in Component Testing?
When and who should do Component testing?
Approach and Responsibilities for Component Testing
Component Testing Tools

What is Unit/Component Testing?

According to ***ISTQB***, ***Component testing*** is the testing of individual hardware or software components. Error detection in these units is simple and less time consuming because the software comprises several units/modules. However, the production of outputs by one unit may become the inputs for another unit. Therefore, if an incorrect output produced by one unit works as an input to the second unit, then it also produces erroneous output. If the first unit contains errors that are not corrected, then all integrating software components may produce unexpected outputs. Therefore, testing of all software units happens independently using *Component testing* to avoid this.

Below are some of the critical considerations of component testing:

Performance of Unit tests of software applications happens during the development (coding) of an application.
The developer usually performs unit tests.
In SDLC, unit tests are the first level of tests performed before integration tests.
Component testing can be a ***WhiteBox or Black-box testing*** *technique that is performed by the developer.* Most of the articles confuse component testing with ***component integration testing***. Component testing happens on a single component/unit. Whereas the component integration test occurs when two components are involved, and one of the components acts as a ***stub*** or a driver.



You can read about component integration testing in detail in our article "***Integration Testing*****.**"

What are the Objectives of Component testing?

Objectives of Component testing include:

- ***Reducing risk***: *Firstly, it verifies every single unit of the application. Developers find out the errors in the code and fix it. Therefore, it reduces the chances of risk at a fundamental level.*
- ***Verifying whether functional and non-functional behaviors of the component are as expected***: *The second objective is to confirm that the functional and non-functional attributes of the component are working correctly. In other words, it ensures that their design and specifications are as expected. It may include functionality (e.g., the correctness of calculations) and non-functional characteristics (e.g., searching for memory leaks).*
- ***Building confidence in the component's quality***: *Thirdly, since the component testing happens on the unit level, most of the errors are detected and removed while coding itself. It develops confidence in the product that it will have a lesser number of errors in further testing.*
- ***Finding defects in the component***: *Its main objective is to find errors in the source code. Moreover, it also verifies functions, control flow, data structure, etc., used in the program.*
- ***Preventing defects from escaping to higher test levels***: *Finally, in Component testing, the coding errors are detected and eliminated by the developers. As a result, it reduces the presence of errors in the higher level of testing.*

Component testing often happens in isolation from the rest of the system. It is mainly dependent on the ***SDLC model*** and the system, which may require ***mock objects, service virtualization, harnesses, stubs, and drivers*****.** Component testing may cover functionality *(e.g., the correctness* *of calculations)*, non-functional characteristics *(e.g., searching for memory leaks)*, and structural properties *(e.g., decision testing)*.

In iterative development models *(E.g., Agile)*, the frequency of builds (*code changes*) is pretty high. As a result, there is always a risk that a new build will break existing functionality. As such component testing becomes vital to ensure that the developer catches any defect in his code before it goes out to the testing team. The best practice is to automate component tests and run them every time the developer checks-in a new code.

What is the Test basis for Component testing?

The ***Test Basis*** is the source of information or the documents needed to write test cases and also for test analysis. The basis of the test must be well defined and adequately structured so that one can quickly identify the test conditions from which the test cases derive.

E.g., for Component testing, the test basis can be as follows:

- *Detailed Design for each component*
- *Code blocks for each component*
- *Data Model that defines how a component will receive data from the upstream component. Along with, how it will pass the data to the down-stream integrating component.*
- *Additionally, it includes Component Specifications that define the architecture of the component.*

What are the Test Objects for Component testing?

Test Object describes what should one test in a test level. It refers to the component, integrated components, or the full system.

For Component testing, test objects can be as follows:

- ***Components, Units or Modules***: *Each component or a unit that a single developer creates should be unit tested by that same developer.*
- ***Code and Data Structures***: *This could include best coding practices, and ensuring that code will not break some other shared component.*
- ***Classes***: *This includes testing each class, and ensuring that correct Object Oriented Principles are put to use. E.g., for a banking application, it is vital to use encapsulation, so another class can not directly access any data in a class. It results in minimizing security threats to the application.*
- ***Database modules***: *A database saves data entered in a User Interface (e.g., A new customer registration). As such, the Database should also undergo testing in component testing along with the front end.*

What are the typical defects and failures in Component testing?

Component tests are used to verify the code produced during software coding, and it is responsible for evaluating the correctness of a particular unit of source code. Typical defects that identified in it are as follows:

Incorrect functionality: *It often results in finding the wrong functionality. E.g., A component that should return the discount value when a customer applies a discount coupon on the Amazon website is not returning any discount value.*
Data Flow Problems: *A component often passes on some data to another integrating component, and this data flow often leads to defects. E.g., the discount component that returns discount value when a customer applies a discount code only accepts alphanumeric characters. The component that creates these discounts is creating discount codes with special characters. On the other hand, if the discount component uses these discount codes, it will lead to data flow problems.*
Incorrect code and logic: *It identifies any issues with the logic. E.g., A* ***"buy two get one free"*** *logic on Amazon site works when three items are in the cart. However, it doesn't work when four things are in the cart.* Defects logged during Component Testing is fixed there and then, and there is no formal defect management process that's followed for unit testing defects. A developer can still log a defect when a root cause analysis is pending, and if the defect is complex and challenging to fix immediately. E.g., A developer is working on a component. However, during component testing, he finds a critical defect which leads to the spillover of his task to the next sprint. In this case, a defect needs to log. Which, in turn, can give visibility to Scrum Master, or Dev Leads about the identification of a critical defect, and pass information that it will take time to get fixed.

When and who should do Component testing?

It happens before Integration testing.
It is the first level of testing performed on the system.
The developers usually do component testing in their local environment before the code propagates to higher settings.
Sometimes, depending on the appropriate risk level, a different programmer performs Component testing, thus introducing independence.

Approach and responsibilities for Component testing

Usually, the developers who write the code perform Component testing. They should do this testing before they move on to develop another component. Once the identification of the defects happen in component testing; either the developer can fix all of them before moving to another component, or he can alternate between the fixing and development alternatively. ***Test-Driven Development (TDD)*** is an example of a test-first approach (*where a test is written first before the development*). Even though the origin of TDD is Extreme Programming, other forms of agile use it as well.

Steps followed in Test Driven Development:

. ***Create a Test***: *In TDD, the first step is to create a failing test case. E.g., if you are creating a login component, you will write a test case which will say that enter a valid user id and password to login successfully.*
. ***Run the test to check that the test fails***: *The test case will fail upon execution because the coding is still incomplete.*
. ***Write the code***: *The developer writes the code to ensure the test case will pass. The objective of code is to pass the test case, and that is the only focus of the code.*
. ***Run the Test***: *If the test case has passed, the developer will move on to the next feature; otherwise, he will move to refactor the code.*
. ***Refactor Code***: *If the test case has failed, the code will need to be modified. In some cases, the test case will pass, but there could be performance issues where code refactoring will happen again.*
. ***Repeat the Cycle***: *The entire process (Step 1-5) repeats until the code refactors to pass the test case.*

Unit Testing Tools

There are several automated tools available to help with Component testing. We will provide some examples below:

Jtest: *Parasoft Jtest accelerates the development of Java software by providing a set of tools like static analysis, unit tests, code coverage, etc. Which, in turn, maximizes quality and minimizes business risks. By automating these time-consuming aspects of unit tests, it frees the developer to focus on business logic and create more meaningful sets of tests.*
Junit: *Junit is an open-source testing tool used for the Java programming language. It provides assertions to identify the test method. This tool first tests the data and then inserts it into the code snippet.*
NUnit: *NUnit is a unit test framework that all .net languages use widely. It is an open-source tool that allows you to write scripts manually. It supports tests based on data that run in parallel.*
JMockit: *JMockit is an open-source Component test tool. It is a code coverage tool with line and route metrics. This tool offers line coverage, route coverage, and data coverage.*
EMMA: *EMMA is an open-source toolkit for analyzing and reporting code written in the Java language. Emma admits coverage types such as method, line, basic block. Its basis is Java. Therefore, it has no external library dependencies and can access the source code.*
PHPUnit: *PHPUnit is a unit test tool for PHP programmers. Take small portions of code called units and test each one separately. The tool also allows developers to use predefined affirmation methods to affirm that a system behaves in a certain way.*

I am sure that by now, you have got a clear understanding of Component testing and its importance in ensuring the overall quality of the product. It ensures that "***Everyone owns the Quality***", and not just the testing team.