Test levels:

Unit testing refers to tests that verify the functionality of a specific section of code, usually at the function level. In an object-oriented environment, this is usually at the class level, and the minimal unit tests include the constructors and destructors.

Unit tests are used to test individual code components and ensure that code works the way it was intended to. Test cases are typically written at a method level.

Integration testing is the phase in software testing in which individual software modules are combined and tested as a group. Integration testing is any type of software testing that seeks to verify the interfaces between components against a software design. Software components may be integrated in an iterative way or all together ("big bang"). Normally the former is considered a better practice since it allows interface issues to be located more quickly and fixed.

Integration Tests check if the system as a whole works. Integration testing is also done by developers, but rather than testing individual components, it aims to test across components. A system consists of many separate components like code, database, web servers, etc. Integration tests are able to spot issues like wiring of components, network access, database issues, etc.

Component interface testing

The practice of component interface testing can be used to check the handling of data passed between various units, or subsystem components, beyond full integration testing between those units. The data being passed can be considered as "message packets" and the range or data types can be checked, for data generated from one unit, and tested for validity before being passed into another unit. Component interface testing is a variation of black-box testing, with the focus on the data values beyond just the related actions of a subsystem component.

Operational Acceptance testing

Acceptance testing is a test conducted to determine if the requirements of a specification or contract are met. It may involve chemical tests, physical tests, or performance tests. This type of testing focuses on the operational readiness of the system to be supported, or to become part of the production environment.

System testing

System testing tests a completely integrated system to verify that the system meets its requirements. For example, a system test might involve testing a logon interface, then creating and editing an entry, plus sending or printing results, followed by summary processing or deletion (or archiving) of entries, then logoff.

Test Driven Development - TDD

Test-driven development (TDD) is a software development process in which tests are written based on the requirements before any coding begins. Since there is no code yet, the test will initially fail. The minimum amount of code is then written to pass the test. The code is then refactored until it is optimized.

Code Coverage - Code coverage measures (in percentage) how much of the code is executed when the unit tests are run.

Mocking

When we learn to program, our objects are usually self contained. Any hello world has no dependencies on outside classes (System.out aside) and neither do many of the other classes we write in the process of learning a language. However, in the real world, software has dependencies. We have action classes that depend on services and services that depend on data access objects (DAOs) and the list goes on.

The idea of unit testing is that we want to test our code without testing the dependencies. The concept behind mock objects is that we want to create an object that will take the place of the real object. This mock object will expect a certain method to be called with certain parameters and when that happens, it will return an expected result. Mock objects provide you with the ability to test what you write without having to address dependency concerns.

Unit Test Case

A Unit Test Case is a part of code, which ensures that another part of code (method) works as expected. To achieve the desired results quickly, a test framework is required. JUnit is a perfect unit test framework for Java programming language.

A formal written unit test case is characterized by a known input and an expected output, which is worked out before the test is executed. There must be at least two unit test cases for each requirement − one positive test and one negative test.

Features of JUnit Test Framework

JUnit test framework provides the following important features −

Fixtures

Test suites

Test runners

JUnit classes

Fixtures is a fixed state of a set of objects used as a baseline for running tests. The purpose of a test fixture is to ensure that there is a well-known and fixed environment in which tests are run so that results are repeatable. It includes −

setUp() method, which runs before every test invocation.

tearDown() method, which runs after every test method.

Test suites bundles a few unit test cases and runs them together. In JUnit, both @RunWith and @Suite annotation are used to run the suite test.

Test runner is used for executing the test cases.

JUnit classes are important classes, used in writing and testing JUnits. Some of the important classes are −

Assert − Contains a set of assert methods.

TestCase − Contains a test case that defines the fixture to run multiple tests.

TestResult − Contains methods to collect the results of executing a test case.

TestSuite A TestSuite is a composite of tests.