**Unit Testing**

* **Why write Unit Test?**
* Make sure that the code works.
* Code works with valid and invalid input parameters.
* Code works now and in the future.
* Other code still works even after you made changes.
* **F.I.R.S.T Principle**

Fast: Unit tests run fast

Independent: Unit tests are independent

Repeatable: Unit tests are repeatable

Self-validating: Unit test validates itself

Thorough & Timely: Cover edge cases

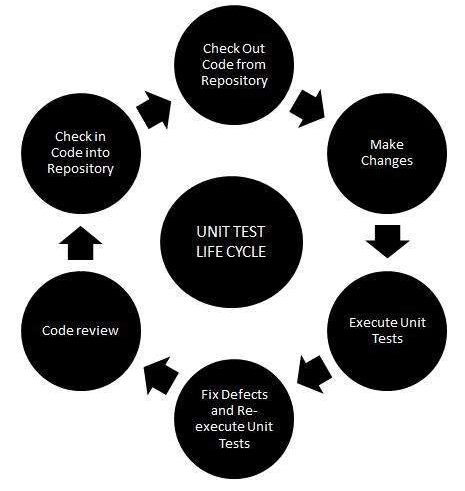
* **Advantages**

Reduce bugs when changing the existing functionality.

Reduces costs of Testing as defects are captured in very early phase.

Improve design and allows better refactoring of code.

* **Unit Testing Lifecycle**



* **Techniques**

Black Box Testing

White Box Testing

Gray Box Testing

* **Testing Pyramid**

1. **Unit tests**

Testing isolated small pieces of code with Fake or Mock dependencies. Run before any other test.

Testing a functionality of a Unit in isolation.

Involves **mock** only the main layer (e.g.: **Web Layer**).

1. **Integration Tests**

Tests use real objects instead of mocks of fake objects. Can interact with database, HTTP connections, servers, communicate with internal or external systems.

Involves **all** **three layers**, so **no mocking** of any layer or any object will be done.

1. **End-To-End Testing / UI Testing**

Testing software functionality from beginning to end. Write automated UI tests. e.g.: invoke button clicks.

* **Test Driven Development - TDD**

Is the process in which test cases are written before the code that validates those cases.

* + TDD Life Cycle

1. Red: write Unit test that fails.
2. Green: write application code to make Unit Test pass.
3. Refactor: Clean up. Improve Unit Test and Application code.
4. Repeat: repeat above steps until all your functionality is implemented.
   * TDD – Three Laws
5. No Production code without failing test.
6. Just enough test to make code fail.
7. Just enough code to make test pass.
   * Best Practice

Don’t create a long list of different assertions in a single method. Have one test method to test one specific functionality and have one strong assertion for it.

<https://www.geeksforgeeks.org/test-driven-development-tdd/>

<https://www.geeksforgeeks.org/test-driven-development-using-junit5-and-mockito/>

\junit\JUnitUtils

* **Data-Driven Testing – DDT**

Is Parameterized testing by applying some conditions like verified inputs.

Types depends of the format of the data: Comma Separated Values (CSV) files, Excel Sheets, Database Tables, Script Arrays, Table Variables.

<https://www.geeksforgeeks.org/difference-between-data-driven-testing-and-keyword-driven-testing/>

* **Automated Software Testing**

<https://www.tutorialspoint.com/software_testing_dictionary/automated_software_testing.htm>

* Functional testing vs Non-Functional testing:

|  |  |
| --- | --- |
| **Functional testing** | **Non-Functional testing** |
| Verifies the **operations** and **actions** of an applications. | Verifies the **behavior** of an application. |
| Based on customer’s **requirements**. | Based on customer’s **expectations**. |
| Based on **business** **requirements**.  Involves: checking of **User interface**, APIs, **Database**, **Security**, **client/server** **applications** and **functionality** of the applications under test. | Based on the **performance** **requirements**., **usability**, **reliability**. |
| Helps to enhance the **behavior** of the application. | Helps to improve the **performance** of the applications. |
| Easy to execute manually or using automation. | Hard to execute manually. |
| Tests **what** the **product** **does**. | Describes **how** the **product** **does**. |
| Examples:   * **Unit Testing** * Smoke Testing * **Integration Testing** * **Regression Testing** * User Acceptance * Localization * Globalization * Interoperability | Examples:   * **Performance Testing** * Load Testing * **Stress Testing** * Scalability Testing * Volume Testing * Usability Testing * Compliance Testing * Portability Testing * Disaster Recover Testing |
| Real example:  Check the **login** functionality. | Real example:  Check the dashboard should load in 2 seconds.  Check **how many people** can simultaneously **login** into a software. |
| Tools:   * Rational Robot (IBM) * Coded UI (Microsoft) * **Selenium** (Open Source) | Tools:   * Load Runner (HP) * **Jmeter** (Apache) |
| Is **performed** **before** the **non-functional testing**. |  |

<https://www.geeksforgeeks.org/differences-between-functional-and-non-functional-testing/>

<https://www.guru99.com/functional-testing-vs-non-functional-testing.html#:~:text=Functional%20testing%20is%20based%20on,the%20performance%20of%20the%20software>.

* **Framework for automated testing for Java**

**JUnit**, **Mockito,** Guava, Spring, **TestNG**

**JUnit** is a standard **unit**-**testing** framework for Java. **JUnit 5** is based in Java 8.

**TestNG** is used for **Integration** testing.

Mocking is a way to test the functionality of a class in isolation.

* **JUnit**
* JUnit 5 (last version 2024)

<https://junit.org/junit5/>

* JUnit Parameterized Test

JUnit 4

Execute the same test case again and again with different values.

<https://www.w3schools.blog/parameterized-test-junit>

* samples

<https://www.geeksforgeeks.org/junit-testcases-for-credit-card-validation-as-a-maven-project/?ref=ml_lbp>

<https://www.geeksforgeeks.org/junit-writing-sample-test-cases-for-studentservice-in-java/>

* **Mockito**

<https://site.mockito.org/>

<https://javadoc.io/doc/org.mockito/mockito-core/latest/org/mockito/Mockito.html>

**Mockito** is an [open source](https://en.wikipedia.org/wiki/Open_source) [testing framework](https://en.wikipedia.org/wiki/List_of_unit_testing_frameworks) for [Java](https://en.wikipedia.org/wiki/Java_(software_platform)) released under the [MIT License](https://en.wikipedia.org/wiki/MIT_License). The framework allows the creation of [test double](https://en.wikipedia.org/wiki/Test_double) objects ([**mock** objects](https://en.wikipedia.org/wiki/Mock_object)) in [automated unit tests](https://en.wikipedia.org/wiki/Test_automation) for the purpose of [test-driven development](https://en.wikipedia.org/wiki/Test-driven_development) (**TDD**) or [**behavior**-driven development](https://en.wikipedia.org/wiki/Behavior-driven_development)(**BDD**).

Mockito also provides some annotations for reducing [boilerplate code](https://en.wikipedia.org/wiki/Boilerplate_code).

Mocks all the methods of that interface (inclusive default methods).

Operations: Mock, Fake, Spy, Stub

<https://www.toptal.com/java/a-guide-to-everyday-mockito#:~:text=With%20Mockito%2C%20you%20create%20a,form%20of%20a%20changed%20state>.

* samples:

<https://www.geeksforgeeks.org/how-to-test-java-list-interface-methods-using-mockito/>

* **Code Coverage**
  + Export Code Coverage Report

In IntelliJ IDEA:

A screenshot of a computer

Description automatically generated

* + Export Test Report using Maven

unit testing\mockito\MockitoUtils

* **Unit Testing and Integration Testing in Spring Boot Application**

\unit testing\springboot

A screen shot of a computer

Description automatically generated

Spring framework allow to test each layer of our application architecture separately.

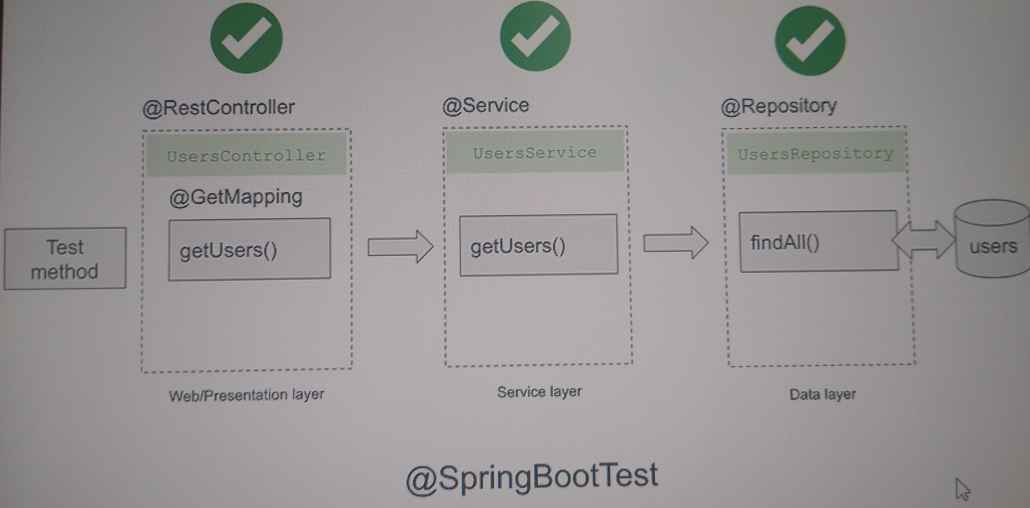
A screenshot of a computer

Description automatically generated

A diagram of a service

Description automatically generated

@SpringBootTest: build the context for testing, by default will not start a web server.



* Automate unit testing generation

Machinet AI: <https://plugins.jetbrains.com/plugin/17510-machinet-ai-gpt-4-chat-and-unit-tests>

* **Courses**
* Testing Java with JUnit 5 & Mockito

<https://tcsglobal.udemy.com/course/testing-java-code-with-junit-5-and-mockito/learn/lecture/31481676#overview>

* **References**

Write test cases in IntelliJ IDEA

<https://www.geeksforgeeks.org/how-to-write-test-cases-in-java-application-using-mockito-and-junit/>

Unit Testing

<https://www.tutorialspoint.com/software_testing_dictionary/unit_testing.htm>

D:\ROMULO\PROJECTS\tech\backend\java\unit testing\utils\JUnitMockitoUtils

D:\ROMULO\PROJECTS\tech\backend\java\springboot\projects\bck\spring-boot-javaguides

Continuous Integration and Continuous Development tools