* **Explain the meaning of Unit testing and its difference on comparison with Functional testing**

**Smallest unit to test mocking dependencies**

Unit testing is a type of software testing where **individual units or components** of a program are tested **in isolation** to ensure they work as expected.

| **Aspect** | **Unit Testing** | **Functional Testing** |
| --- | --- | --- |
| Scope | Individual methods/classes | End-to-end features or use cases |
| Granularity | Very small (single method/function) | Large (whole module/page/form) |
| Dependencies | Mocked or stubbed | Real dependencies (DB, API) used |
| Who writes it | Developers | Testers or QA |
| Tools | NUnit, MSTest, xUnit | Selenium, Postman, Cypress |
|  |  |  |

Unit Testing often uses **mocking** to simulate external dependencies like databases or APIs.

* **List various types of testing**
  + **Unit testing, Functional testing, Automated testing, Performance testing**

· **Unit Testing**: Test small code units in isolation.

· **Functional Testing**: Check the system against requirements.

· **Integration Testing**: Verify that modules work together.

· **System Testing**: Validate the system as a whole.

· **Regression Testing**: Ensure new changes don’t break existing code.

· **Automated Testing**: Use tools to run tests automatically.

· **Performance Testing**: Check how the system performs under load.

· **Acceptance Testing**: Verify the app meets business needs.

* **Understand the benefit of automated testing**

Fast feedback during development

Reduces manual testing time

Ensures code stability on every change

Helps with continuous integration and delivery (CI/CD)

Improves confidence in refactoring code

Repeatable and consistent

* **Explain what is loosly coupled & testable design**
  + **Write code that is NOT dependent on the class for data.**

· **Loosely Coupled Design**: Components are independent and communicate via interfaces.

· **Testable Design**: Code is written in a way that makes it easy to write tests.

**Bad Example:**

var db = new RealDatabase();

**Good Example (loosely coupled):**

public class Service

{

private readonly IDataStore \_dataStore;

public Service(IDataStore dataStore)

{

\_dataStore = dataStore;

}

}

* **Write your first testing program to validate a calculator addition operation**

**TestFixture, Test**

[TestFixture]

public class CalculatorTests

{

[Test]

public void Test\_Addition()

{

var calc = new SimpleCalculator();

double result = calc.Addition(2, 3);

Assert.That(result, Is.EqualTo(5));

}

}

* **Understand the need of [SetUp], [TearDown] & [Ignore] attributes.**

| **Attribute** | **Use** |
| --- | --- |
| [SetUp] | Runs **before every test**. Used to initialize objects. |
| [TearDown] | Runs **after every test**. Used for cleanup. |
| [Ignore] | Skips the test method temporarily. |

* **Explain the benefit of writing parameterised test cases.**

**TestCase**

[Test]

[TestCase(1, 2, 3)]

[TestCase(10, 5, 15)]

[TestCase(-1, -1, -2)]

public void Test\_Addition\_WithVariousInputs(double a, double b, double expected)

{

var calc = new SimpleCalculator();

Assert.That(calc.Addition(a, b), Is.EqualTo(expected));

}

**Benefits:**

Less code

Easily test multiple scenarios

Improves test coverage