Software testing ensures that applications function correctly and meet requirements. Here’s a **brief introduction** to different types of testing and unit testing:

**Types of Software Testing**

1. **Unit Testing** – Tests individual components or functions in isolation to verify correctness.
2. **Integration Testing** – Ensures different modules work together properly.
3. **Functional Testing** – Validates that the software meets business requirements.
4. **System Testing** – Tests the entire system as a whole.
5. **Acceptance Testing** – Confirms the software meets user expectations.
6. **Performance Testing** – Evaluates speed, responsiveness, and stability.
7. **Security Testing** – Identifies vulnerabilities and ensures data protection.

**Unit Testing Overview**

Unit testing focuses on testing **small, self-contained pieces of code**, such as functions or methods, to ensure they work as expected. It helps:

* Detect bugs early in development.
* Improve code quality and maintainability.
* Support **Test-Driven Development (TDD)** practices.
* Reduce debugging time and costs.

Unit tests can be **manual** or **automated**, with frameworks like **NUnit** for .NET or **pytest** for Python. They typically follow a **white-box testing approach**, meaning they test internal logic rather than just inputs and outputs.

**Custom attributes in NUnit** allow developers to extend the framework by defining their own attributes for test cases. NUnit uses attributes extensively to control test execution, parameterization, and setup.

**Key Concepts of Custom Attributes in NUnit**

* NUnit provides standard interfaces for custom attributes, categorized by their role in the **test life cycle**.
* **Load-Time Interfaces** help define test fixtures and parameterized test cases.
* **Execution-Time Interfaces** modify test behavior during execution.

**Examples of Custom Attributes**

1. **IFixtureBuilder** – Defines how a test fixture is created.
2. **ITestBuilder** – Generates parameterized test cases.
3. **IApplyToTest** – Modifies test properties after creation.
4. **IApplyToContext** – Sets up the test execution context.

To **configure a project with NUnit assemblies**, follow these steps:

**1. Install NUnit Packages**

* Open your project in **Visual Studio**.
* Use **NuGet Package Manager** to install:
* NUnit
* NUnit3TestAdapter
* Microsoft.NET.Test.Sdk

**2. Create a Test Project**

* Add a new **Class Library (.NET)** project.
* Reference the NUnit package in the test project.

**3. Write a Basic NUnit Test**

using NUnit.Framework;

[TestFixture]

public class SampleTests

{

[Test]

public void TestExample()

{

Assert.AreEqual(4, 2 + 2);

}

}**4. Configure Test Runner**

* Ensure NUnit3TestAdapter is installed for **Test Explorer** integration.
* Run tests using **Test Explorer** in Visual Studio.