**WEEK – 2 Assignments**

**NUnit and Moq**

**Mandatory hands-on :-**

**1. NUnit-Handson**

**Objectives :-**

## 1.Unit Testing vs Functional Testing

Unit testing validates individual code components (like methods or functions) in isolation by mocking dependencies. It focuses on internal logic correctness at the smallest level.

Functional testing evaluates end-to-end system behavior from a user perspective, using real dependencies to verify overall functionality.

Key differences:

* Unit tests are narrow in scope (single components), while functional tests cover broader workflows.
* Unit tests mock dependencies for isolation; functional tests use real implementations.
* Unit tests are developer-centric and fast; functional tests are user-centric and slower.

## 2.Types of Testing

1. Unit testing: Isolated component validation.
2. Functional testing: End-to-end feature verification.
3. Automated testing: Scripted test execution.
4. Performance testing: System responsiveness/scalability checks.

## 3.Benefits of Automated Testing

* Accelerates test execution and feedback cycles.
* Reduces human error in repetitive tasks.
* Enables continuous integration/deployment.
* Lowers long-term maintenance costs.
* Improves test coverage consistency.

## 4.Loosely Coupled & Testable Design

Loosely coupled code minimizes dependencies between components, allowing independent modification and testing. Testable design avoids hardcoding dependencies:

* Use abstractions (interfaces) instead of concrete classes.
* Inject dependencies externally rather than instantiating them internally.
* Ensure classes don’t control their own data sources.

## 5.Writing a Calculator Test

Steps to validate addition:

1. Create a test class marked with [TestFixture].
2. Declare the calculator object.
3. Initialize the object in a [SetUp] method.
4. Write a test method with [TestCase] for parameterized inputs.
5. Use Assert.That to compare actual and expected results.
6. Cleanup resources in [TearDown].

## 6.Purpose of Attributes

* [SetUp]: Prepares pre-test environment (e.g., initializing objects).
* [TearDown]: Releases post-test resources (e.g., closing connections).
* [Ignore]: Temporarily skips tests without deleting code.

## 7.Benefits of Parameterized Tests

* Eliminates code duplication for similar test scenarios.
* Centralizes test data management.
* Simplifies edge-case testing.
* Enhances readability by grouping related cases.

**TestFixture & Test :-**

**Codes :-**

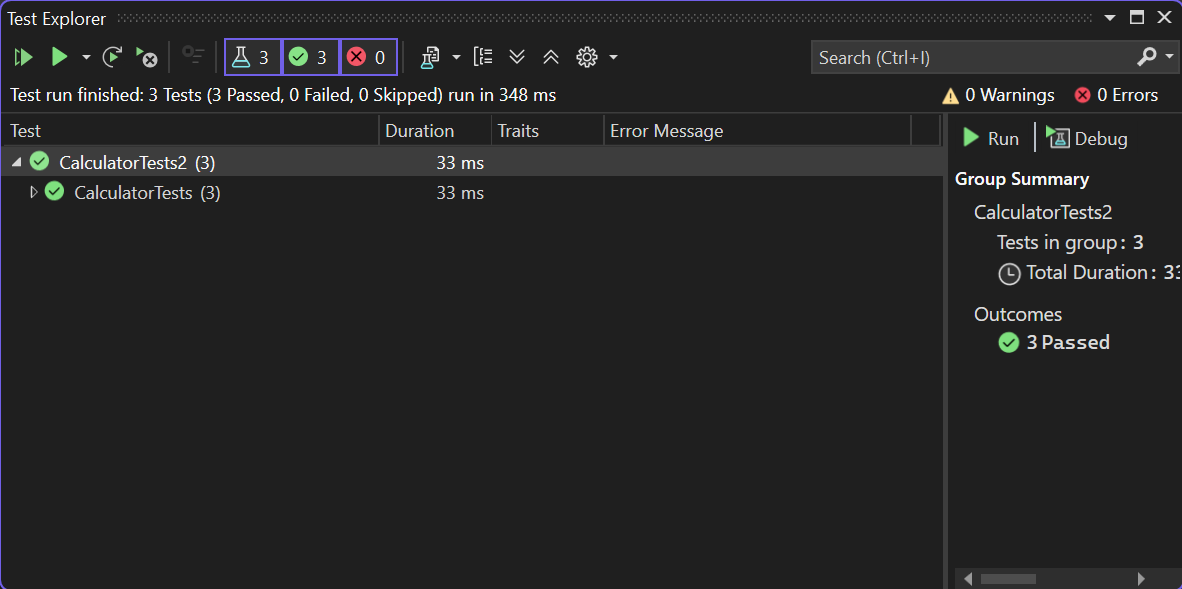
**Clac1.cs :-**

| namespace CalcLibrary {  public class Calculator  {  public int Add(int a, int b) => a + b;  public void Dispose() { /\* Cleanup \*/ }  } } |
| --- |

**UnitTest1 :-**

| using NUnit.Framework; using CalcLibrary;  namespace CalculatorTests {  [TestFixture]  public class CalculatorTests  {  private Calculator \_calc;   [SetUp]  public void Setup()  {  \_calc = new Calculator();   }   [TearDown]  public void Cleanup()  {  \_calc.Dispose();   }   [TestCase(2, 3, 5)]  [TestCase(-1, 5, 4)]  [TestCase(0, 0, 0)]  public void TestAddition(int a, int b, int expected)  {  // Act  int result = \_calc.Add(a, b);   // Assert  Assert.That(result, Is.EqualTo(expected));  }  } } |
| --- |

**Test Results :-**

****