**OBJECTIVES:**

**Unit Testing:**

->Tests the **smallest possible piece of code** — usually a **single method** or function.

->It **isolates** that unit and checks if it works **correctly**.

->It often uses **mocking** to fake dependencies, so you only test the **unit** — not other parts.

**Functional Testing:**

->Tests **complete features** or **end-to-end scenarios**.

->Checks whether the **system behaves** as expected based on **requirements**

->Does **not isolate units** — it tests the **flow** through multiple units working together.

**Key difference:**  
 Unit test: Small, isolated piece -> developer-level  
 Functional test: Full feature -> user/business-level

## 2 ****Various Types of Testing****

**->Unit Testing:** Test single code units (methods/classes).

**->Functional Testing:** Test business features or workflows.

**->Automated Testing:** Run tests automatically using tools/scripts.

**->Performance Testing:** Check speed, load, and scalability under stress.

## 3 ****Benefit of Automated Testing****

->Saves time by **running tests automatically** every time you change code.

->Catches **bugs early** → better quality.

->Supports **Continuous Integration (CI/CD)** pipelines.

->Gives developers confidence to refactor without breaking things.

## 4 first Unit Test: Calculator Addition

**CODE (EXAMPLE):-**

**using NUnit.Framework;**

**[TestFixture]**

**public class CalculatorTests**

**{**

**[Test]**

**public void Add\_WhenCalled\_ReturnsSum()**

**{**

**// Arrange**

**var calc = new Calculator();**

**// Act**

**int result = calc.Add(2, 3);**

**// Assert**

**Assert.AreEqual(5, result);**

**}**

**}**

**public class Calculator**

**{**

**public int Add(int a, int b)**

**{**

**return a + b;**

**}**

**}**

## 5 ****Loosely Coupled & Testable Design****

**Loosely Coupled:**  
Design your code so **classes and methods depend on interfaces**, not concrete classes → easy to **mock** dependencies in tests.

**Testable Design:**  
E.g., a class should **not directly fetch data itself** → instead, it should get data through **injected interfaces** (like repositories/services)

## ****6 Need for [SetUp],[TearDown] and [Ignore]****

[SetUp] → Runs **before every test**. Good for initializing common objects.

[TearDown] → Runs **after every test**. Good for cleanup, like closing files.

[Ignore] → Temporarily skip a test. E.g., if it’s broken or depends on something not ready yet.

## 7 ****Benefit of Parameterized Tests ([TestCase])****

**->Write one test method** → run it with **multiple inputs & expected outputs**.

->Avoids duplicate code.

->Makes it easier to cover **multiple scenarios** with minimal code.

**TestFixture & Test**

**Using Already Given CalcLibrary:**

**MathLibrary.cs CODE:-**

**using System;**

**namespace CalcLibrary**

**{**

**interface IMathLibrary**

**{**

**double Addition(double a, double b);**

**double Subtraction(double a, double b);**

**double Multiplication(double a, double b);**

**double Division(double a, double b);**

**}**

**public class SimpleCalculator : IMathLibrary**

**{**

**double result = 0;**

**public double Addition(double a, double b)**

**{**

**result = a + b;**

**return result;**

**}**

**public double Subtraction(double a, double b)**

**{**

**result = a - b;**

**return result;**

**}**

**public double Multiplication(double a, double b)**

**{**

**result = a \* b;**

**return result;**

**}**

**public double Division(double a, double b)**

**{**

**if (b == 0)**

**throw new ArgumentException("Second Parameter Can't be Zero");**

**result = a / b;**

**return result;**

**}**

**public void AllClear()**

**{**

**result = 0;**

**}**

**public double GetResult**

**{**

**get { return result; }**

**}**

**}**

**}**

**CalculatorTests.cs CODE:-(We need to create it using dotnet)**

**using NUnit.Framework;**

**using CalcLibrary;**

**namespace CalcLibraryTests**

**{**

**[TestFixture]**

**public class CalculatorTests**

**{**

**private SimpleCalculator \_calculator;**

**[SetUp]**

**public void Setup()**

**{**

**\_calculator = new SimpleCalculator();**

**}**

**[TearDown]**

**public void Teardown()**

**{**

**\_calculator.AllClear();**

**\_calculator = null;**

**}**

**[Test]**

**[TestCase(2, 3, 5)]**

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

**[TestCase(0, 0, 0)]**

**public void Addition\_Returns\_Correct\_Result(double a, double b, double expected)**

**{**

**double result = \_calculator.Addition(a, b);**

**Assert.That(result, Is.EqualTo(expected));**

**}**

**[Test]**

**[TestCase(5, 3, 2)]**

**[TestCase(0, 0, 0)]**

**public void Subtraction\_Returns\_Correct\_Result(double a, double b, double expected)**

**{**

**double result = \_calculator.Subtraction(a, b);**

**Assert.That(result, Is.EqualTo(expected));**

**}**

**[Test]**

**[TestCase(2, 3, 6)]**

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

**public void Multiplication\_Returns\_Correct\_Result(double a, double b, double expected)**

**{**

**double result = \_calculator.Multiplication(a, b);**

**Assert.That(result, Is.EqualTo(expected));**

**}**

**[Test]**

**[TestCase(6, 3, 2)]**

**public void Division\_Returns\_Correct\_Result(double a, double b, double expected)**

**{**

**double result = \_calculator.Division(a, b);**

**Assert.That(result, Is.EqualTo(expected));**

**}**

**[Test]**

**public void Division\_By\_Zero\_Throws\_Exception()**

**{**

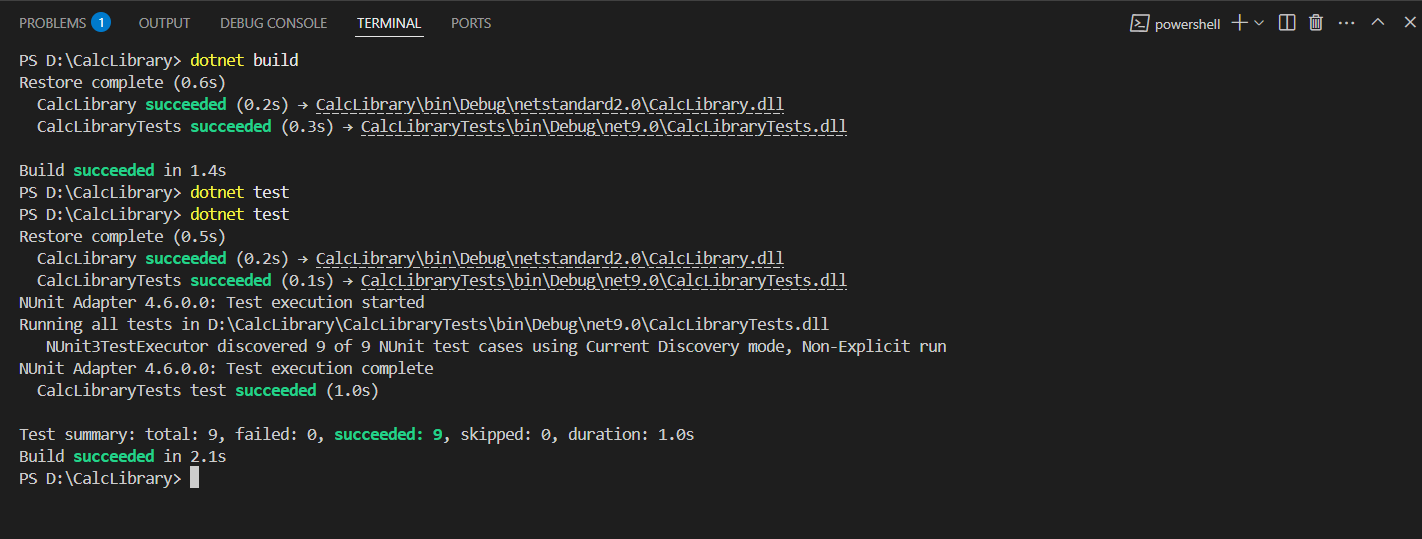
**Assert.Throws<System.ArgumentException>(() => \_calculator.Division(10, 0));**

**}**

**}**

**}**

**OUTPUT:-**



**OBJECTIVES:-**

1. **Write Testable Code with Moq:**

**(We have to Create CustomerCommLib using dotnet)**

**CustomerCommLib:**

**MailSender.cs (Code):-**

**using System.Net;**

**using System.Net.Mail;**

**namespace CustomerCommLib**

**{**

**public interface IMailSender**

**{**

**bool SendMail(string toAddress, string message);**

**}**

**public class MailSender : IMailSender**

**{**

**public bool SendMail(string toAddress, string message)**

**{**

**MailMessage mail = new MailMessage();**

**SmtpClient SmtpServer = new SmtpClient("smtp.gmail.com");**

**mail.From = new MailAddress("your\_email\_address@gmail.com");**

**mail.To.Add(toAddress);**

**mail.Subject = "Test Mail";**

**mail.Body = message;**

**SmtpServer.Port = 587;**

**SmtpServer.Credentials = new NetworkCredential("username", "password");**

**SmtpServer.EnableSsl = true;**

**SmtpServer.Send(mail);**

**return true;**

**}**

**}**

**}**

**CustomerComm.cs (Code):-**

**namespace CustomerCommLib**

**{**

**public class CustomerCommu**

**{**

**IMailSender \_mailSender;**

**public CustomerCommu(IMailSender mailSender)**

**{**

**\_mailSender = mailSender;**

**}**

**public bool SendMailToCustomer()**

**{**

**\_mailSender.SendMail("cust123@abc.com", "Some Message");**

**return true;**

**}**

**}**

**}**

**CustomerComm.Tests:**

**CustomerCommTests.cs (Code):-**

**using NUnit.Framework;**

**using Moq;**

**using CustomerCommLib;**

**namespace CustomerComm.Tests**

**{**

**[TestFixture]**

**public class CustomerCommTests**

**{**

**private Mock<IMailSender> \_mailSenderMock;**

**private CustomerCommu \_customerComm;**

**[OneTimeSetUp]**

**public void Init()**

**{**

**\_mailSenderMock = new Mock<IMailSender>();**

**\_mailSenderMock.Setup(x => x.SendMail(It.IsAny<string>(), It.IsAny<string>())).Returns(true);**

**\_customerComm = new CustomerCommu(\_mailSenderMock.Object); // <-- renamed**

**}**

**[Test]**

**public void SendMailToCustomer\_WhenCalled\_ReturnsTrue()**

**{**

**bool result = \_customerComm.SendMailToCustomer();**

**Assert.That(result, Is.True);**

**}**

**} }**

**OUTPUT:-**

