**Week2:**

**Nunit And Moq:**

**1.Nunit HandsOn:**

**1.· Explain the meaning of Unit testing and its difference on comparison with Functional testing. Smallest unit to test mocking dependencies**

**Ans:**

Unit testing is a software testing method where the smallest testable parts of an application (units, such as functions or methods) are tested in isolation to ensure they work as intended.

* The smallest unit to test is typically a single function or method.
* *Mocking dependencies* means simulating external components (like databases or APIs) so the unit can be tested independently.

Difference between Unit Testing and Functional Testing:

| Aspect | Unit Testing | Functional Testing |
| --- | --- | --- |
| Scope | Smallest units (methods/functions/classes) | Complete features/user flows |
| Isolation | Yes, uses mocks | No, tests integrated components |
| Focus | Internal logic | End-to-end system behavior |

**2.· List various types of testing:**

**o Unit testing, Functional testing, Automated testing, Performance testing**

**Ans:**

Types of Testing:

1. Unit Testing  
   Tests individual components or functions of an application in isolation to ensure each part works as intended.
2. Functional Testing  
   Verifies that specific features or functions of the software operate according to requirements, often by simulating user actions.
3. Automated Testing  
   Uses software tools to automatically execute tests, compare actual outcomes with expected results, and report findings, enabling faster and more reliable testing.
4. Performance Testing  
   Assesses how a system performs under various conditions, such as heavy load or high usage, to ensure responsiveness, stability, and scalability.

**3.· Understand the benefit of automated testing**

**Ans:**

## Benefits of Automated Testing:

* Early detection of bugs
* Faster development and deployment cycles
* Safer code refactoring
* Documentation of intended code behavior
* Consistent and repeatable test execution

**4.· Explain what is loosly coupled & testable design**

**Ans:**Loosely coupled design means that software components are independent and interact with each other through well-defined interfaces, not through direct dependencies on each other’s internal details. This makes the system more flexible, scalable, and easier to maintain because changes in one component do not require changes in others.

Testable design refers to writing code in a way that makes it easy to test individual components in isolation. This is achieved by minimizing dependencies between components and using interfaces or abstractions, so mock objects can be used during testing

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

**Ans:**

* [SetUp]:  
   Runs before each test to initialize resources.
* [TearDown]:  
  Runs after each test to clean up resources.
* [Ignore]:  
  Skips the marked test, so it is not executed.

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

Parameterized test cases let you execute the same test logic with different input values, helping you verify your code works under a variety of conditions. This approach increases test coverage by checking multiple scenarios in a single test, prevents code duplication, and makes your test suite easier to maintain and expand as your code evolves.

**TestFixture & Test:**

**Code:**

[**calculator.cs**](http://calculator.cs)**:**

namespace CalcLibrary

{

public class Calculator

{

public int Add(int a, int b)

{

return a + b;

}

}

}

[**CalculatorTest.cs**](http://calculatortest.cs)**:**

**using NUnit.Framework;**

**using CalcLibrary;**

**namespace CalcLibrary.Tests**

**{**

**[TestFixture]**

**public class CalculatorTests**

**{**

**private Calculator \_calculator;**

**[SetUp]**

**public void SetUp()**

**{**

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

**}**

**[TearDown]**

**public void TearDown()**

**{**

**\_calculator = null;**

**}**

**[Test]**

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

**{**

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

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

**}**

**[TestCase(1, 2, 3)]**

**[TestCase(-1, 1, 0)]**

**[TestCase(100, 200, 300)]**

**public void Add\_TestCases(int a, int b, int expected)**

**{**

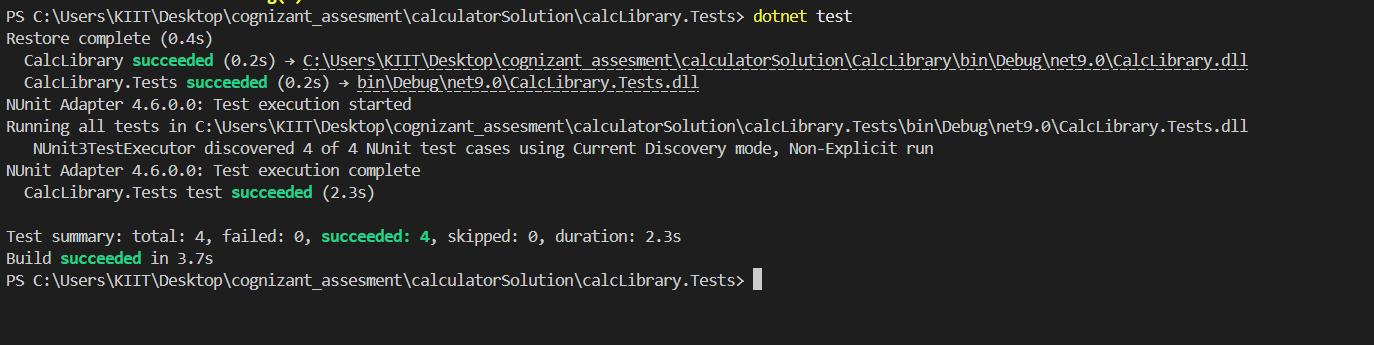
**int result = \_calculator.Add(a, b);**

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

**}**

**}}**

**Output:**

****

**2.Moq Hands-On:**

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

**Task1: In this task, you will create a class library that will be used for unit testing.**

**Codes:**

**CustomerCommLib:**

[**MailSender.cs**](http://mailsender.cs)**:**

**using System.Net;**

**using System.Net.Mail;**

**namespace CustomerCommLib**

**{**

**public class MailSender : IMailSender**

**{**

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

**{**

**try**

**{**

**using (var mail = new MailMessage("your\_email@gmail.com", toAddress))**

**using (var smtp = new SmtpClient("smtp.gmail.com", 587))**

**{**

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

**mail.Body = message;**

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

**smtp.EnableSsl = true;**

**smtp.Send(mail);**

**}**

**return true;**

**}**

**catch**

**{**

**return false;**

**}**

**}**

**}**

**}**

[**IMailSender.cs**](http://imailsender.cs)**:**

**namespace CustomerCommLib**

**{**

**public interface IMailSender**

**{**

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

**}**

**}**

[**CustomerComm.cs**](http://customercomm.cs)**:**

**namespace CustomerCommLib**

**{**

**public class CustomerComm**

**{**

**private readonly IMailSender \_mailSender;**

**public CustomerComm(IMailSender mailSender)**

**{**

**\_mailSender = mailSender;**

**}**

**public bool SendMailToCustomer()**

**{**

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

**}**

**}**

**}**

**CustomerCommLib.Tests:**

[**CustomerCommTest.cs**](http://customercommtest.cs)**:**

**using CustomerCommLib;**

**using Moq;**

**using NUnit.Framework;**

**namespace CustomerCommLib.Tests**

**{**

**[TestFixture]**

**public class CustomerCommTests**

**{**

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

**private CustomerComm \_customerComm;**

**[SetUp]**

**public void Setup()**

**{**

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

**\_customerComm = new CustomerComm(\_mockMailSender.Object);**

**}**

**[Test]**

**public void SendMailToCustomer\_ShouldReturnTrue\_WhenMailSucceeds()**

**{**

**\_mockMailSender.Setup(m => m.SendMail(It.IsAny<string>(), It.IsAny<string>()))**

**.Returns(true);**

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

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

**}**

**[Test]**

**public void SendMailToCustomer\_ShouldCallSendMailOnce()**

**{**

**\_mockMailSender.Setup(m => m.SendMail(It.IsAny<string>(), It.IsAny<string>()))**

**.Returns(true);**

**\_customerComm.SendMailToCustomer();**

**\_mockMailSender.Verify(**

**m => m.SendMail(It.IsAny<string>(), It.IsAny<string>()),**

**Times.Once**

**);**

**}**

**[Test]**

**public void SendMailToCustomer\_ShouldUseCorrectEmailAndMessage()**

**{**

**const string expectedEmail = "cust123@abc.com";**

**const string expectedMessage = "Some Message";**

**\_mockMailSender.Setup(m =>**

**m.SendMail(expectedEmail, expectedMessage))**

**.Returns(true);**

**\_customerComm.SendMailToCustomer();**

**\_mockMailSender.Verify(**

**m => m.SendMail(expectedEmail, expectedMessage),**

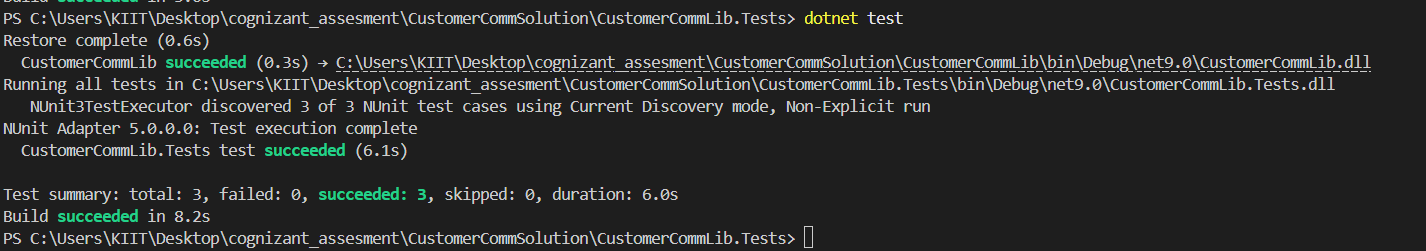
**Times.Once**

**);**

**}**

**}**

**}**

**Output:**

**Task2: In this task, you will create unit test project which make use of NUnit framework and Moq.**

**Code:**

**CustomerComm.Tests:**

[**CustomerCommTests.cs**](http://customercommtests.cs)**:**

**using NUnit.Framework;**

**using Moq;**

**using CustomerCommLib;**

**namespace CustomerCommLib.Tests**

**{**

**[TestFixture]**

**public class CustomerCommTests**

**{**

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

**private CustomerComm \_customerComm;**

**[OneTimeSetUp]**

**public void Init()**

**{**

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

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

**\_customerComm = new CustomerComm(\_mockMailSender.Object);**

**}**

**[TestCase]**

**public void SendMailToCustomer\_ShouldReturnTrue()**

**{**

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

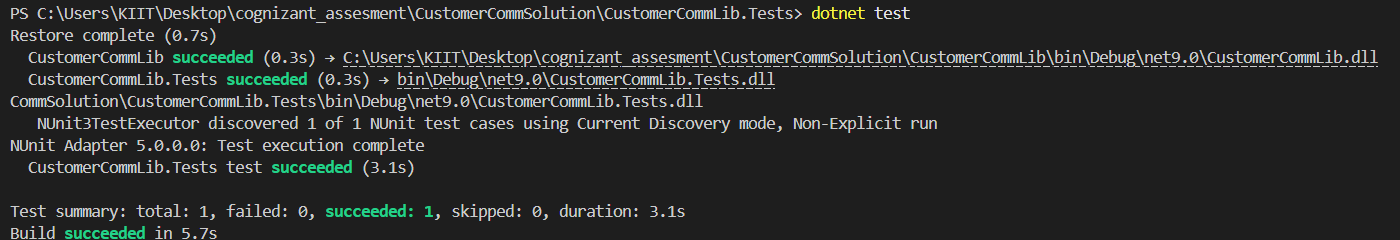
**Assert.That(result, Is.True);**

**}**

**}**

**}**

**Output:**

****