Name:-Adnan Hasan

GitHub Link:-

[https://github.com/adnan-984/Cognizant\_6364530\_Adnan](#_top)CognizantDotNet

**-----------------------------------------------------------------------------------------**

**NUnit  
  
Exercise 1:   
  
Objectives**

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

**Unit Testing** means testing the **smallest unit of code**, like a method or function, to make sure it works as expected.

In unit testing, we often **mock dependencies** (like databases or services) so that we only test that small piece of logic in isolation.

**Functional Testing** means testing a complete feature or function of the application to see if it meets the requirements.

Functional testing checks the software from the **user’s point of view**, without focusing on the internal code.

Smallest unit to test mocking dependencies  
  
Ans:- In **Unit Testing**, we test the **smallest part of the code** ,usually a **method or function**.Sometimes, that method depends on **other classes.**Instead of using the real database or service, we use a **mock,** so that we can test the method **independently**.

* List various types of testing

1. Unit testing, Functional testing, Automated testing, Performance testing  
     
     
     
     
   Ans:-

**Unit Testing**

Tests the smallest part of the application (like a function or method).

Done by developers.

Example: Testing a calculator's addition method.

**Functional Testing**

Tests the complete functionality of the software based on requirements.

Checks what the system **does**, not how.

Example: Testing if login works with valid credentials.

**Automated Testing**

Tests are written and executed by tools automatically.

Saves time and effort in repetitive testing.

Tools: Selenium, NUnit, JUnit, etc.

**Performance Testing**

Checks how fast or stable the system is under load.

Helps identify bottlenecks.

Types: Load testing, stress testing.

Understand the benefit of automated testing  
  
Ans:- Automated testing helps test the software quickly without doing it manually every time. It saves time and effort, especially when we have to test the same features again and again. It also reduces human errors and improves the quality of the software. Automated tests can run anytime, even at night, which helps in faster development and delivery. Over time, it becomes more cost-effective and supports continuous integration and deployment.

* Explain what is loosly coupled & testable design

Ans:- A **loosely coupled** design means that different parts of the code are independent and not tightly connected. This makes it easier to **test, maintain, and reuse** the code. In a **testable design**, we avoid hardcoding dependencies inside a class. Instead, we pass them from outside (like using constructor injection or interfaces). This allows us to **mock or replace dependencies** during testing.

Write code that is NOT dependent on the class for data.

Ans:- public interface IPaymentService

{

void MakePayment();

}

public class PaymentService : IPaymentService

{

public void MakePayment()

{

Console.WriteLine("Payment done.");

}

}

public class OrderService

{

private IPaymentService \_payment;

public OrderService(IPaymentService payment)

{

\_payment = payment;

}

public void ProcessOrder()

{

\_payment.MakePayment();

}

}

* Write your first testing program to validate a calculator addition operation
  + TestFixture, Test

Ans:- **Calculator Class**  
  
  
public class Calculator

{

public double Addition(double a, double b)

{

return a + b;

}

}  
  
**Unit Test using NUnit  
  
using NUnit.Framework**

namespace CalcLibraryTests

{

[TestFixture]

public class CalculatorTests

{

[Test]

public void TestAddition()

{

Calculator calc = new Calculator();

double result = calc.Addition(5, 3);

Assert.AreEqual(8, result);

}

}

}

#### Understand the need of [SetUp], [TearDown] & [Ignore] attributes. Ans:- [SetUp]

This method runs **before every test**.

#### Used to initialize objects or set common pre-conditions. [TearDown]

This method runs **after every test**.

Used to clean up, dispose objects, or reset data.

#### [Ignore]

This is used to **skip a test** temporarily (maybe under development or failing due to known reason).

* Explain the benefit of writing parameterised test cases.

TestCase  
  
Ans:- **Parameterized test cases** allow you to test **multiple sets of inputs** and **expected outputs** using a **single test method**.  
This makes your test code **cleaner**, **shorter**, and **easier to manage**.

**Calculator.cs**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 Calculator : 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**using NUnit.Framework;

using CalcLibrary;

namespace CalcLibraryTests

{

[TestFixture]

public class CalculatorTests

{

private Calculator calc;

[SetUp]

public void Setup()

{

calc = new Calculator();

}

[TearDown]

public void Teardown()

{

calc = null;

}

[Test]

[TestCase(5.0, 3.0, 8.0)]

[TestCase(10.5, 4.5, 15.0)]

[TestCase(-1.0, -2.0, -3.0)]

public void TestAddition(double a, double b, double expected)

{

double result = calc.Addition(a, b);

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

}

}

}  
  


