COGNIZANT DN - 4.0 DEEP SKILLING

HANDS ON WEEK-2

NUnit and Moq

**OBJECTIVES:**

1.Explain the meaning of Unit testing and its difference on comparison with Functional testing

Smallest unit to test mocking dependencies.

**SOLUTION:**

**Unit Testing** is a software testing method where you test the **smallest testable parts of an application** (called **units**) in isolation to ensure they work as expected.

**Functional Testing** tests the **entire feature or function** of an application based on the **business requirements**.

2. List various types of testing

**SOLUTION:**

**1. Unit Testing**

* **Definition**: Testing individual units/components of a program.
* **Goal**: Ensure each function or method works as intended.

### 2. ****Functional Testing****

* **Definition**: Testing based on the system's functionality.
* **Goal**: Check if the system behaves as expected for given inputs.

### 3. ****Automated Testing****

* **Definition**: Using scripts/tools to execute tests automatically.
* **Goal**: Save time and improve consistency in testing.

### 4. ****Performance Testing****

* **Definition**: Testing the speed, responsiveness, and stability under load.
* **Subtypes**: Load testing, stress testing, scalability testing.

3. Explain what is loosly coupled & testable design.

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

**SOLUTION:**

### **Loosely Coupled Design**

A design where **components are independent** of each other — changes in one class **do not affect** others. This improves **flexibility**, **maintainability**, and **testability**.

* Loosely Coupled = Less dependency between classes.
* Strongly Coupled = Hard to modify or test.

### **Testable Design**

A design that makes it **easy to write unit tests** by:

* Using interfaces or abstractions
* Avoiding hardcoded dependencies
* Injecting dependencies (e.g., via constructors)

**CODE:**

using System;

public interface IEmailService

{

void SendEmail(string to);

}

public class EmailService : IEmailService

{

public void SendEmail(string to)

{

// Real email sending logic

Console.WriteLine($"Email sent to {to}");

}

}

public class ReportGenerator

{

private readonly IEmailService \_emailService;

// Injecting dependency

public ReportGenerator(IEmailService emailService)

{

\_emailService = emailService;

}

public void GenerateReport()

{

// Report generation logic

\_emailService.SendEmail("report@example.com");

}

}

**4.** Write your first testing program to validate a calculator addition operation.

**CODE:**

using System;

public interface IEmailService

{

void SendEmail(string to);

}

public class EmailService : IEmailService

{

public void SendEmail(string to)

{

// Real email sending logic

Console.WriteLine($"Email sent to {to}");

}

}

public class ReportGenerator

{

private readonly IEmailService \_emailService;

// Injecting dependency

public ReportGenerator(IEmailService emailService)

{

\_emailService = emailService;

}

public void GenerateReport()

{

// Report generation logic

\_emailService.SendEmail("report@example.com");

}

}

5. Explain the benefit of writing parameterised test cases.

**SOLUTION:**

Here are the benefits of writing parameterised test case:

 **Avoids Repetition**: You don't need to write the same test code again and again for different inputs.

 **Saves Time**: You can test many cases by writing just one method with different input values.

 **Improves Test Coverage**: Allows testing with various inputs, including edge cases.

 **Easier to Maintain**: If the test logic changes, you only need to update it in one place.

 **Keeps Code Clean**: Reduces the number of test methods and keeps the test class organized.