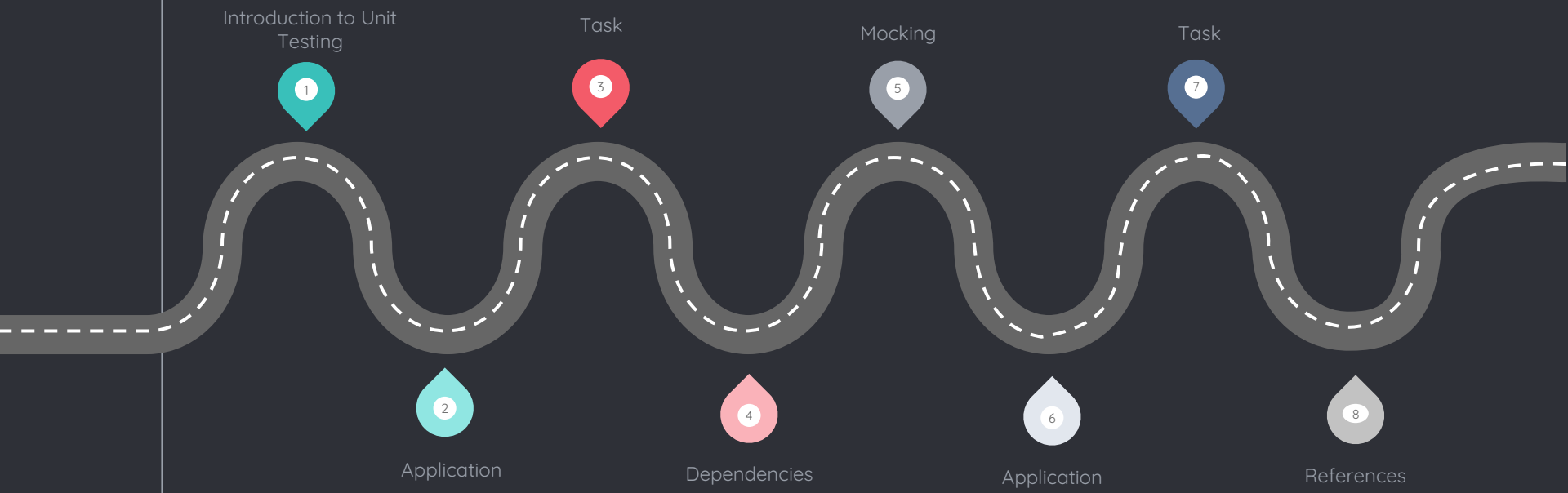
Unit Tests

Waleed Elwakeel

Course Roadmap



1

Introduction to Unit Testing

MSTest

1.1 What is Unit tests?

Unit tests are low-level tests, focusing on a small part of the software system.

1.2 Types of Testing:

1 Unit testing :

Test specific function only. Test cases only are used

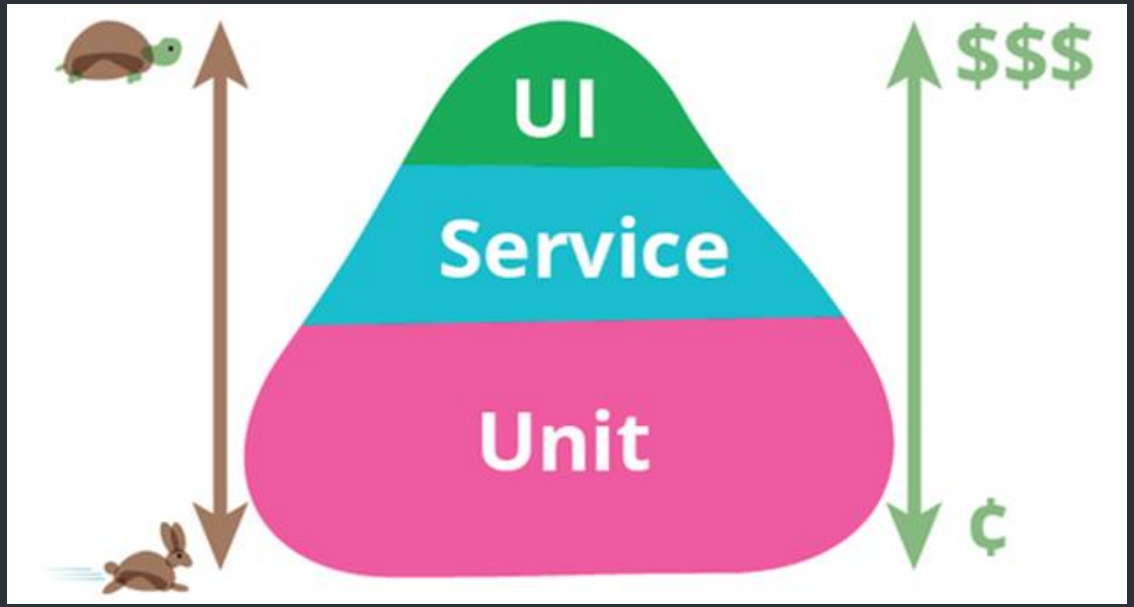
2 Integration Testing:

Test multiple behaviors together, test scenarios

3 Acceptance testing:

Done by the client before delivering.

1.3 Test Pyramid:



1.4 Why do we use unit Testing?

1) Validates the functionality of the code. For example, test the behavior, test the data type.
2) Allows you to add new feature/service or modify it without affecting the rest of the code Unintentionally.
3) It is more time saving and efficient than using manual testing.

2 Application

2.1 Naming Convention

- **Class:**
  Name of the class + tests ([className]Tests)
  example: student.cs ➔ studentTests.cs
- **Methods:**
  MethodName_StateUnderTest_ExpectedBehavior
  (admitStudent_IfMandatoryFieldsAreMissing_FailToAdmit)

2.2 Testing Pattern

Public void method with **[TestMethod]** attribute on the function, and **[TestClass]** Attribute on the class.

Usually AAA (Arrange – Act – Assert) pattern is used.

Public void TestMethodName(){

// Arrange
Preparing the needed objects and dependencies.

// Act
Execute the method/behavior.

//Assert

Make sure the output behavior is the expected behavior.
}

2.3 Rules of unit testing

- Test each function independently.
- It has one path (no If / else)
- Doesn’t depend on other functions.
- Avoid logic in tests
- Using clear convention (Naming – testing pattern)

2.4 Assert class.

- Assert is sealed class uses static methods which is used to validate the required tests.

Types of Assert Classes:

1) Assert.
2) StringAssert
3) CollectionAssert

2.4.1 Assert

Used to validate single objects.

Function Name	Usage
AreEqual	Assert if both values are equal.
AreNotEqual	Assert if both values are not equal.
AreSame	Assert if both objects are same.
AreNotSame	Assert if both objects are not same.
IsInstanceOfType	Assert if it is the type of the object checked
IsNotInstanceOfType	Assert if it is not the type of the object checked
IsNull	Assert if it is null
IsNotNull	Assert if it is not null
IsTrue	Assert if condition is true
IsFalse	Assert if condition is false

2.4.2 StringAssert

Used to validate strings.

Function Name	Usage
Contains	Assert if string contains the substring
DoesNotMatch	Assert if string does not match regex
Matches	Assert if string matches regex
EndsWith	Assert if string ends with substring
StartsWith	Assert if string starts with substring

2.4.3 CollectionAssert

Used to validate collection of objects.

Function Name	Usage
AllItemsAreInstancesOfType	Assert if the collection is of the same type
AllItemsAreNotNull	Assert if the collection items don't contain nulls
AllItemsAreUnique	Assert if the collection items are unique
AreEqual	Assert if the collection is equal to another one, equality here having the same elements in the same order and quantity. Different collection references to the same value are considered equal.
AreNotEqual	Assert if the collection is not equal to another one
AreEquivalent	Assert if the collection contains the same elements as the other collection.
AreNotEquivalent	Assert if the collection doesn't contain the same elements as the other collection.
Contains	Assert if the collection contains certain element
DoesNotContain	Assert if the collection doesn't contain certain element
IsSubsetOf	Assert if the collection is subset of super set
IsNotSubsetOf	Assert if the collection is not subset of super set

2.5 Attributes

They are added on the function to provide some functionalities.

Attribute	Usage
ExpectedException(typeof(ExceptionType))]	Test if it will throw the expected exception.
AssemblyInitialize	Called once before the project starts
AssemblyCleanUp	Called once before the project ends
ClassInitialize	Called once before the Class starts
ClassCleanup	Called once before the class ends
TestInitialize	Called once before each test starts
TestCleanup	Called once before each test ends
Owner	Group by owner
Priority	Group by priority
TestCategory	Group by Test category
Ignore	Ignore this method

3

Task

3.1 Required Tasks

- 4 Unit tests using Assert class
- 2 Unit tests using StringAssert Class
- 2 Unit tests using CollectionAssert Class

4 Dependencies

Dependencies

Whenever a class can't perform its behavior without the use of another class then there is a dependency between them. Dependency leads to coupling between the classes, the tighter the coupling gets, the more hard the class can be reused and increase maintenance.

Dependency injection is used to **decouple** the usage of the object from its creation, which decreases the coupling between depenedencies leading it to become loose.

Dependency Injection (DI)

Dependency Injection is a software design technique in which the creation and binding of dependencies are done outside of the dependent class.

Dependency Inversion Principle (DIP)

- High-level modules should not depend on low-level modules. Both should depend on abstractions.
- Abstractions should not depend on details. Details should depend on abstractions.

Inversion of Control (IoC)

Inversion of control is a programming principle that inverts the flow of control in an application. In traditional procedural programming, the code that controls the execution of the program — the main function — instantiates objects, calls methods and even asks the user for input so that the execution can continue and the program can achieve its task. With IoC, it is a framework that does the instantiation, method calls and triggers user actions, having full control of the flow and removing this responsibility from the main function, and by consequence the application.

5 Mocking

Mocking

Mocking is used to generate fake dependencies during testing, which makes the tests isolated and not brittle.

Steps of Mocking using MOQ:

1. Create mock of the dependency.
2. Build the mock data.
3. Setup the called method
4. Use the fake object as dependency

6 Application

7

Task

7.1 Required Tasks

- 3 Test Methods on Repositories classes (using MOQ).
- 3 Test Methods on Services classes (using MOQ).

8 References

References

- https://martinfowler.com/bliki/UnitTest.html
- https://martinfowler.com/bliki/TestPyramid.html
- https://martinfowler.com/articles/practical-test-pyramid.html#UnitTests
- https://docs.microsoft.com/en-us/dotnet/core/testing/unit-testing-best-practices
- https://medium.com/ssense-tech/dependency-injection-vs-dependency-inversion-vs-inversion-of-control-lets-set-the-record-straight-5dc818dc32d1
- https://dzone.com/articles/difference-between-black-box-white-box-and-grey-bo
- https://www.youtube.com/playlist?list=PLwj1YcMhLRN28xijrXMO255JHsO3csus-

Thanks!

ANY QUESTIONS?

You can find me at

linkedin.com/in/waleedelwakeel

Waleed_elwakeel13@hotmail.com

0100-019-2312