**Доклад**

**по**

**Софтуерно инженерство**

**Тема: Unit Tests**

1. **Unit test**
2. **Регресивно тестване**
3. **Mocking при тестване**
4. **Интеграционно тестване**
5. **Софтуерна документация**
6. **Източници**

Изготвил: Ваня Ванева 12а

1. **Unit test**

Unit тестовете са създадени с цел да проверят дали отделните части от софтуерния код работят правилно, независимо от останалата част от системата. Те трябва да бъдат лесни за изпълнение и бързи за написване. Използват се специализирани инструменти и библиотеки за автоматизиране на изпълнението на тези тестове.

Пример: Нека имаме прост клас **Calculator**, който извършва математически операции:

**public class Calculator**

**{**

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

**{**

**return a + b;**

**}**

**}**

Създаване на unit тест за този клас с използване на библиотеката за тестване NUnit:

**using NUnit.Framework;**

**[TestFixture]**

**public class CalculatorTests**

**{**

**[Test]**

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

**{**

**Calculator calculator = new Calculator();**

**int result = calculator.Add(3, 5);**

**Assert.AreEqual(8, result);**

**}**

**}**

1. **Регресивно тестване**

Регресивното тестване е процесът на проверка дали промените в кода не са повлияли на вече работещите функционалности на приложението. Unit тестовете играят важна роля в регресивното тестване, като гарантират, че промените не са довели до нарушаване на съществуващата функционалност.

Пример: Ако направим промяна в горния пример, като добавим нов метод към класа **Calculator**:

**public class Calculator**

**{**

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

**{**

**return a + b;**

**}**

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

**{**

**return a - b;**

**}**

**}**

Регресивен тест за да проверим дали тази промяна не е повлияла на съществуващите функционалности:

**[Test]**

**public void Subtract\_ValidInput\_ReturnsDifference()**

**{**

**Calculator calculator = new Calculator();**

**int result = calculator.Subtract(8, 3);**

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

**}**

1. **Mocking при тестване**

Mocking е техника, при която се създават фалшиви обекти или модули, които имитират поведението на реалните компоненти в системата. Тази техника е полезна, когато тестваме части от системата, които зависят от външни ресурси или услуги, които може да бъдат трудни за контролиране или трудни за достъпване по време на изпълнение на тестове.

Пример: как можем да използваме Moq за mocking на външна зависимост:

**using Moq;**

**using NUnit.Framework;**

**public interface IExternalService**

**{**

**int GetData();**

**}**

**public class MyClass**

**{**

**private IExternalService externalService;**

**public MyClass(IExternalService externalService)**

**{**

**this.externalService = externalService;**

**}**

**public int PerformAction()**

**{**

**// Някаква логика, използваща външна услуга**

**return externalService.GetData();**

**}**

**}**

**[TestFixture]**

**public class MyClassTests**

**{**

**[Test]**

**public void PerformAction\_WithMockedService\_ReturnsExpectedResult()**

**{**

**var mockService = new Mock<IExternalService>();**

**mockService.Setup(x => x.GetData()).Returns(42);**

**MyClass myClass = new MyClass(mockService.Object);**

**int result = myClass.PerformAction();**

**Assert.AreEqual(42, result);**

**}**

**}**

1. **Интеграционно тестване**

Интеграционното тестване се фокусира върху проверката на взаимодействието между различните компоненти на системата. В този контекст, unit тестовете могат да бъдат използвани за да се гарантира, че всяка интеграция между компонентите работи правилно.

Пример как можем да направим интеграционен тест за два класа, които си взаимодействат:

**using NUnit.Framework;**

**public class ClassA**

**{**

**public int GetNumber()**

**{**

**return 10;**

**}**

**}**

**public class ClassB**

**{**

**private ClassA classA;**

**public ClassB(ClassA classA)**

**{**

**this.classA = classA;**

**}**

**public int GetNumberFromA()**

**{**

**return classA.GetNumber();**

**}**

**}**

**[TestFixture]**

**public class IntegrationTests**

**{**

**[Test]**

**public void ClassB\_GetNumberFromA\_ReturnsExpectedResult()**

**{**

**ClassA classA = new ClassA();**

**ClassB classB = new ClassB(classA);**

**int result = classB.GetNumberFromA();**

**Assert.AreEqual(10, result);**

**}**

**}**

Имаме два класа, ClassA и ClassB, където ClassB използва ClassA. В тестовият клас създаваме инстанция на ClassA и ClassB и тестваме дали методът GetNumberFromA() на ClassB връща очакваната стойност, която е взета от ClassA.

1. **Софтуерна документация**

Unit тестовете могат да послужат и като част от софтуерната документация. Предоставянето на примерни тестове може да помогне на другите разработчици да разберат как работи дадена функционалност и как да я използват правилно.

Пример как може да изглежда такава документация с пример за unit тест:

**/// <summary>**

**/// Клас, който представя базова математическа операция за събиране.**

**/// </summary>**

**public class Calculator**

**{**

**/// <summary>**

**/// Извършва събиране на две числа.**

**/// </summary>**

**/// <param name="a">Първото число.</param>**

**/// <param name="b">Второто число.</param>**

**/// <returns>Сборът на двете числа.</returns>**

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

**{**

**return a + b;**

**}**

**}**

**/// <summary>**

**/// Unit тестове за класа Calculator.**

**/// </summary>**

**[TestFixture]**

**public class CalculatorTests**

**{**

**/// <summary>**

**/// Тест за метода Add, който проверява дали връща правилния резултат при валидни входни данни.**

**/// </summary>**

**[Test]**

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

**{**

**// Arrange**

**Calculator calculator = new Calculator();**

**// Act**

**int result = calculator.Add(3, 5);**

**// Assert**

**Assert.AreEqual(8, result);**

**}**

**// Други тестове за различни сценарии могат да бъдат добавени тук...**

**}**

1. **Източници**

Час по теория

<https://www.toptal.com/qa/how-to-write-testable-code-and-why-it-matters>

<https://www.guru99.com/unit-testing-guide.html>

<https://www.headspin.io/blog/regression-testing-a-complete-guide>

<https://www.browserstack.com/guide/regression-testing>

<https://www.softwaretestingstuff.com/what-is-mock-testing>

<https://www.codium.ai/blog/mock-testing/>

<https://www.techtarget.com/searchsoftwarequality/definition/integration-testing>

<https://brightsec.com/blog/unit-testing/>

<https://aws.amazon.com/what-is/unit-testing/#:~:text=Unit%20testing%20is%20the%20process,test%20for%20each%20code%20unit>.