Министерство образования и науки

Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики

Факультет инфокоммуникационных технологий

Кафедра программных систем

Отчет

по лабораторной работе № 8

# «Организация тестирования»

**Система управления проектами**

по дисциплине «Архитектура программного обеспечения инфокоммуникационных систем»

**Выполнил: студент группы K4220**

**Проверил: к.т.н., доцент Н.А. Осипов**

Санкт-Петербург

2017

## Цель работы.

– освоение процесса тестирования реализации архитектуры прикладных программных систем, разработки функциональных моделей прикладных программных систем,

– управление требованиями к программной системе" (Requirements) в соответствии с Rational Unified Process.

## Задачи, решаемые при выполнении работы.

### Создание тестов программ реализации архитектуры приложения.

### Определить элементы, отображаемые на схемах.

## Объект исследования.

Архитектура приложения.

## Метод экспериментального исследования.

Тестирование.

## Рабочие формулы и исходные данные.

Платформа Unit Testing Framework.

## Схема работы.

### Создание тестирующего проекта

Модульный тест последовательно выполняется для каждой строки в источнике данных. Это облегчает тестирование разнообразных входных данных в ходе одного тестового запуска.

Настройка модульного теста для извлечения значений из источника данных включает два этапа. Сначала необходимо создать соединение между методом модульного теста и источником данных с использованием свойств модульного теста. Второй этап - присвоить переменные метода, так чтобы извлекать их значения из соответствующего столбца источника данных. Это выполняется с применением логики самого метода теста.

## Окончательные результаты.

### Разработка теста.

Тестовый метод. Метод должен проверять результат, полученный для определенных входных данных:

[TestMethod]

public void BasicRooterTest()

{

// Create an instance to test:

Rooter rooter = new Rooter();

// Define a test input and output value:

double expectedResult = 2.0;

double input = expectedResult \* expectedResult;

// Run the method under test:

double actualResult = rooter.SquareRoot(input);

// Verify the result:

Assert.AreEqual(expectedResult, actualResult,

delta: expectedResult / 100);

}.

#### Выполнение теста



Выполните модульный тест.

В меню Тест выберите Запуск - Все тесты в решении.

Просмотрите информацию о результатах выполнения теста.

#### Оптимизация кода

После создания кода программы последним этапом является оптимизация кода, имеющей целью упростить код.

Оптимизация – это изменение, которое делает код более производительным или более понятным. Она не предназначена для изменения поведения кода, поэтому тесты не должны изменяться.

Рекомендуется выполнять действия по оптимизации отдельно от шагов, которые расширяют функциональные возможности. Неизменяемость тестов уменьшает шансы случайных ошибок во время оптимизации.

В цикле упростите расчет переменной результата следующим образом:

public class Rooter

{

public double SquareRoot(double input)

{

...

result = (result + input / result) / 2;

return result;

}

}

Запустите все тесты.

### Использование модульных тестов для улучшения кода

В этой части упражнения описывается, как последовательный процесс анализа, разработки модульных тестов и рефакторинга может помочь сделать рабочий код более надежным и эффективным.

Анализ проблем

После создания тестового метода, проверяющего корректное списывание нужного значение в методе Debit, остались еще два варианта исходного анализа:

Метод создает исключение ArgumentOutOfRangeException, если размер кредита превышает баланс.

Метод создает исключение ArgumentOutOfRangeException, если размер кредита меньше нуля.

### Создание тестовых методов

1. Добавим в файл BankAccountTests.cs первый вариант создания тестового метода для решения второй проблемы:

//unit test method

[TestMethod]

[ExpectedException(typeof(ArgumentOutOfRangeException))]

public void Debit\_WhenAmountIsLessThanZero\_ShouldThrowArgumentOutOfRange()

{

// arrange

double beginningBalance = 11.99;

double debitAmount = -100.00;

BankAccount account = new BankAccount("Mr. Bryan Walton", beginningBalance);

// act

account.Debit(debitAmount);

// assert is handled by ExpectedException

}

Атрибут ExpectedExceptionAttribute был использован для подтверждения правильности возникающего исключения.

Данный атрибут приводит к тому, что тест завершается неудачей, если не возникает исключения ArgumentOutOfRangeException.

## Выводы и анализ результатов работы.

Платформа Unit Testing Framework поддерживает выполнение модульных тестов в Visual Studio. При написании кода модульных тестов используются классы и члены пространства имен Microsoft.VisualStudio.TestTools.UnitTesting. Они используются как при создании модульных тестов с нуля, так и при доработке тестов, созданных из проверяемого кода.