Министерство науки и высшего образования Российской Федерации

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ
(ТУСУР)

Кафедра компьютерных систем в управлении и проектировании

Отчет

Лабораторная работа по курсу новых технологий в программировании

«Юнит-тестирование»

Преподаватель: Студент гр. 588-1:

Горяинов А. Е. _____ Дудик Н. А. _____

Дата: « » _____20__ г. Дата: « » _____20__ г.

2022 г.

Содержание

Введение

Целью данной лабораторной работы является изучение организации тестирования в разработке ПО и получение навыков в написании юнит-тестов.

1 Описание и выполнение задания

Юнит-тестирование – тестирование минимальных модулей архитектуры, изолированных друг от друга. В ООП таковыми являются классы. Юнит-тесты составляются обычно на основе разработанных алгоритмов, а не на основе технического задания.

В рамках данной лабораторной работы требуется создать проект юнит-тестов, рассчитать цикломатическую сложность классов бизнес-логики и покрыть классы юнит-тестами.

UML-диаграмма классов с указанными цикломатическими сложностями выглядит следующим образом.

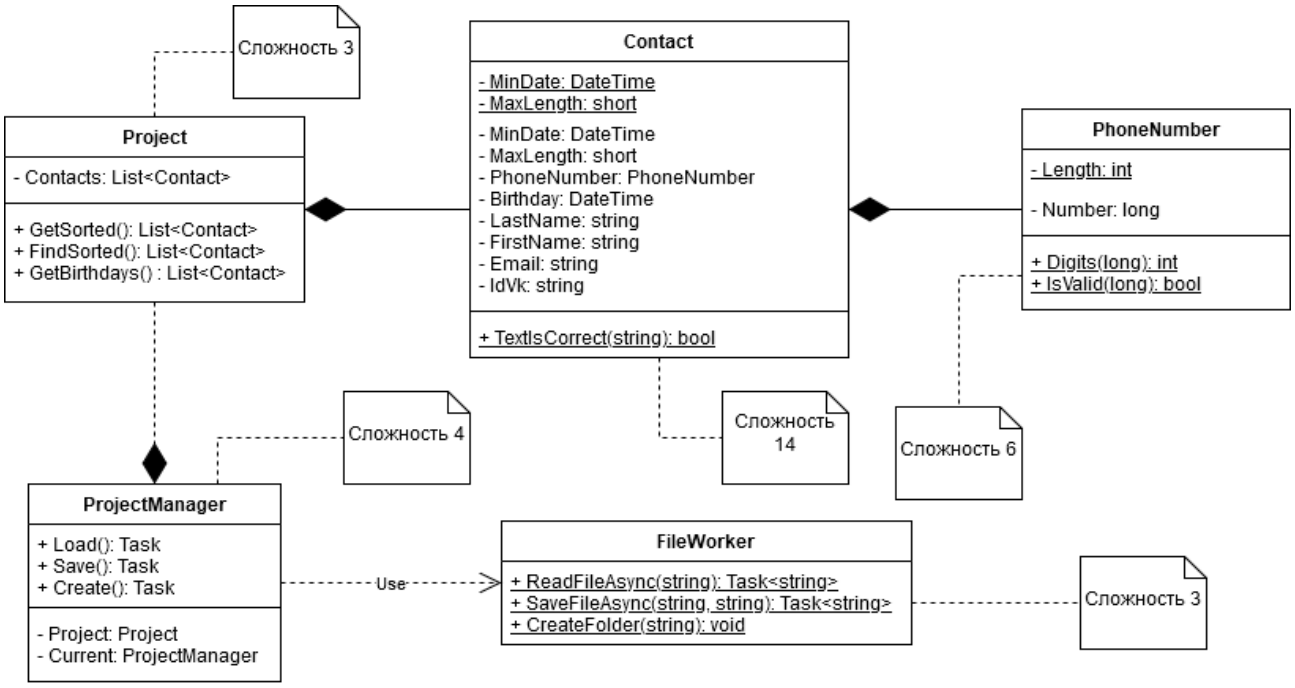

Рисунок 1 UML-диаграмма классов бизнес-логики с цикломатическими сложностями

С помощью Тестирования в Microsoft Visual Studio можно определить степень покрытия кода бизнес-логики юнит-тестами.

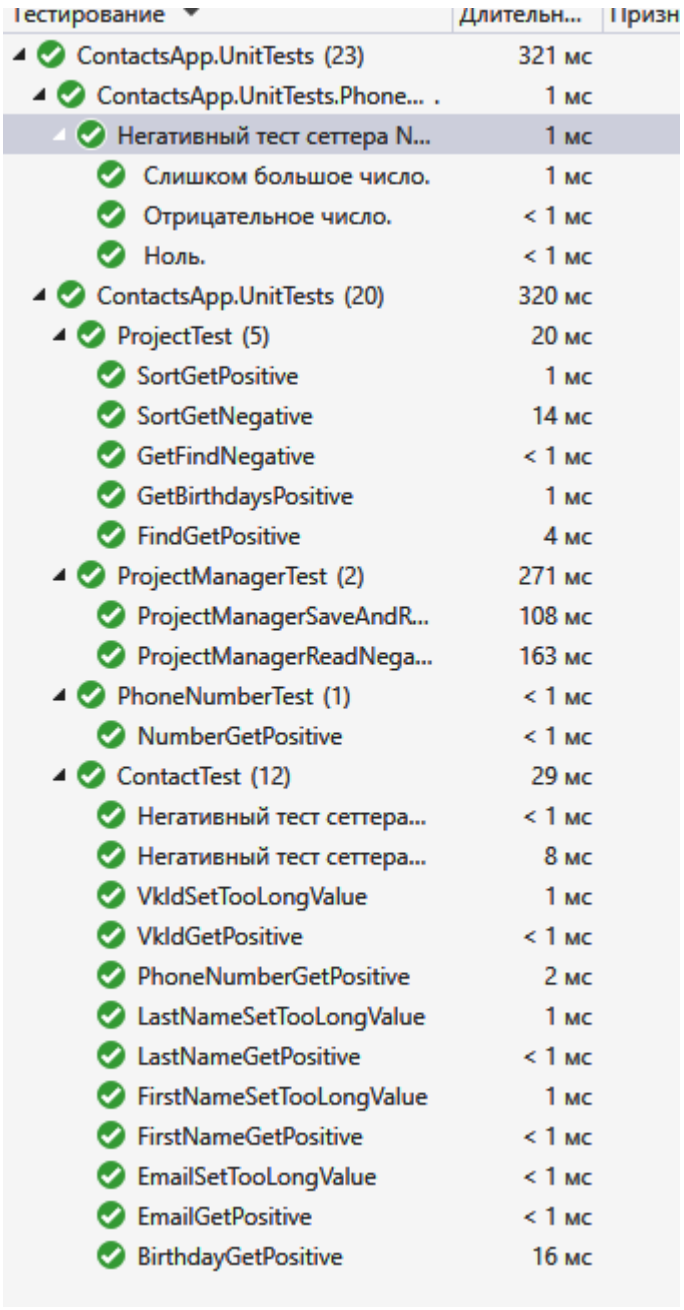

Рисунок 2 Результаты юнит-тестов

В качестве примера юнит-теста рассмотрим класс PhoneNumberTest.cs. Класс теста должен иметь атрибут [TestFixture].

```
[TestFixture]
public class PhoneNumberTest
{...}
```

Перед непосредственным началом тестов выполняются методы, помеченные атрибутом SetUp. В данном случае инициализируется объект PhoneNumber в поле класса.

```
[SetUp]
public void Init()
{
    _phoneNumber = new PhoneNumber();
}
```

Сами тестовые методы должны быть помечены атрибутами Test или TestCase. Позитивный тест геттера свойства Number имеет один тестовый случай и поэтому помечен атрибутом Test. Цель: установить значение свойства, считать его и убедиться, что оно соответствует ожидаемому. Сам тест выглядит следующим образом.

```
[Test(Description = "Позитивный тест геттера Number.")]
public void NumberGetPositive()
       {
        var num = 78005553535;
        _phoneNumber.Number = num;
        Assert.AreEqual(num,_phoneNumber.Number, "Геттер Number вернул неверное значение.");
       }
```

Методы класса Assert проверяют результат теста и используются для обозначения успешного или неудачного прохождения тестового случая. В данном случае использован метод AreEqual, сравнивающий два объекта.

Негативный тест сеттера подразумевает проверку правильной работы механизма валидации данных. В нашем случае номер телефона должен состоять из 11 цифр и начинаться с 7. Разработаны четыре тестовых случая, при которых сеттер свойства Number должен выдать ArgumentException. Каждый тестовый случай помечается атрибутом [TestCase].

```
[TestCase("1111111112323233",
            "Присвоение слишком большого значения.",
            TestName = "Негативный тест сеттера Number. Слишком большое число.")]
[TestCase("1",
            "Присвоение слишком маленького значения.",
            TestName = "Негативный тест сеттера Number. Слишком большое число.")]
[TestCase("-100",
            "Присвоение отрицательного значения.",
            TestName = "Негативный тест сеттера Number. Отрицательное число.")]
[TestCase("0",
            "Присвоение нуля.",
            TestName = "Негативный тест сеттера Number. Ноль.")]
[Test(Description = "Негативный тест сеттера Number.")]
public void NumberSetNegative(string num, string message)
{
var phone = long.Parse(num);
Assert.Throws<ArgumentException>(() => { _phoneNumber.Number = phone; });
}
```

В случае присвоения некорректного значения номера телефона будет брошено исключение ArgumentException, и тест будет считаться пройденным только в этом случае.

```
[Test(Description = "Number validation test: correct value")]
public void TestValidation_CorrectNumber()
{
    long val = 79008007000;
    Assert.IsTrue(PhoneNumber.IsValid(val));
}
```

Данный тест будет считаться успешным, если переданное в метод IsTrue() класса Assert булево значение будет истинным. Конкретно здесь проверяется правильность работы метода проверки номера телефона на корректность.

Всего написано 23 юнит-теста. Время работы каждого из них приведено ниже.

1. TestBirthdaySet_TooLate (2 случая, 11 мс всего)
2. TestBirthdayGet_CorrectValue (8 мс)
3. TestEmailGet_CorrectValue (5 мс)
4. TestEmailSet_TooLongValue (3 мс)
5. TestFirstNameGet_CorrectValue (2 мс)
6. TestFirstNameSet_TooLongValue (2 мс)
7. TestIdVkGet_CorrectValue (3 мс)
8. TestIdVkSet_TooLongValue (3 мс)
9. TestLastNameGet_CorrectVa lue (4 мс)
10. TestLastNameSet_TooLongValue (3 мс)
11. TestPhoneNumberGet_CorrectValue (4 мс)
12. TestNumberSet_IncorrectNumber (4 случая, 28 мс всего)
13. TestValidation_IncorrectNumbers (2 случая, 17 мс всего)
14. TestNumberGet_CorrectValue (4 мс)
15. TestValidation_CorrectNumber (3 мс)
16. FileWorkerTest_CreateFolder (17 мс)
17. FileWorkerTest_WriteProject (37 мс)
18. ProjectManagerTest_ReadCorruptedFile (16 мс)
19. ProjectManagerTest_SaveAndLoad (42 мс)
20. ProjectTest_FindNothingTest (6 мс)
21. ProjectTest_GetBirthdaysList (5 мс)

Все тесты пройдены успешно. Общее время составило 321 мс.

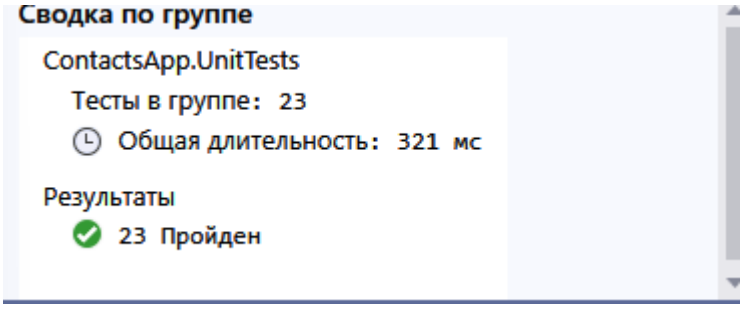

Рисунок 3 Отчёт Visual Studio о прохождении тестов

История коммитов в ветке develop на момент создания данного отчёта выглядит следующим образом.

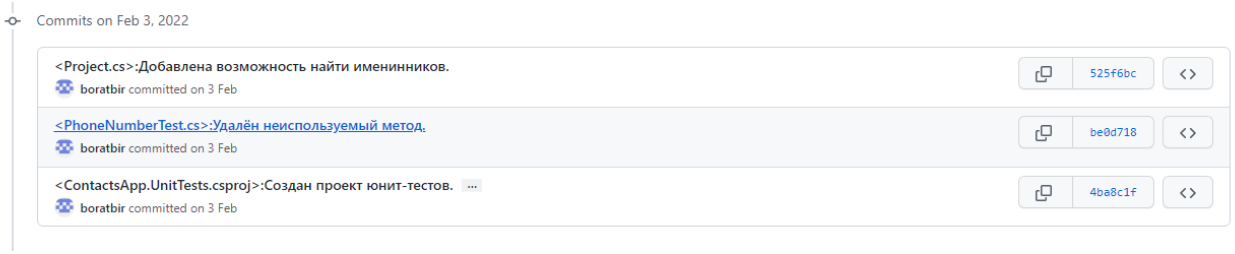

Рисунок 4 История коммитов

2 Выводы

В результате выполнения данной лабораторной работы была изучена организация тестирования в разработке ПО, а также получены навыки в написании юнит-тестов.