Лабораторная работа №5

“Исследование способов модульного тестирования программного

обеспечения в среде NUnit”

1 ЦЕЛЬ РАБОТЫ

Исследовать эффективность использования методологии TDD при разработке программного обеспечения. Получить практические навыки использования фреймворка NUnit для модульного тестирования программного обеспечения.

2 ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. Реализовать на языке C# один из классов, спроектированных в лабораторной работе № 1. Методы класса при этом не реализовывать.
2. Разработать для созданного класса набор модульных тестов, включающий тесты для каждого метода.
3. Запустить набор тестов, проанализировать и сохранить результаты.
4. Поочередно реализовать методы класса, выполняя тестирование при каждом изменении программного кода.
5. После того, как весь набор тестов будет выполняться успешно, реализацию классов можно считать завершенной.

3 ТЕКСТ ПРОГРАММЫ

Файл Symbols.cs:

using System;

namespace SymbolsLib

{

public class Symbols

{

private string str;

public int countForwardSlash = 0;

public int countExclamationMark = 0;

public Symbols(string \_str)

{

str = \_str;

}

public Tuple<int, int> CheckAndPrint()

{

Check(str);

Console.WriteLine("Количество символов / в строке: " + countForwardSlash);

Console.WriteLine("Количество символов ! в строке: " + countExclamationMark);

return new Tuple<int, int>(countForwardSlash, countExclamationMark);

}

public void Check(string str)

{

foreach (var ch in str)

{

if (ch == '/')

++countForwardSlash;

else if (ch == '!')

++countExclamationMark;

}

}

}

}

Файл SymbolsTest.cs:

using System;

using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace SymbolsLib.Tests

{

[TestClass]

public class SymbolsTests

{

readonly string[] strings = {

"Тест",

"!",

"/",

"Тест /!/ тест",

"Тест!",

"!/Тест",

"///!//!!!!/////!!!!!!!!!!!!!!///!/"

};

readonly Tuple<int, int>[] results = {

Tuple.Create(0, 0),

Tuple.Create(0, 1),

Tuple.Create(1, 0),

Tuple.Create(2, 1),

Tuple.Create(0, 1),

Tuple.Create(1, 1),

Tuple.Create(14, 20)

};

[TestMethod]

public void TestNumber1()

{

for (int i = 0; i < strings.Length; i++)

{

Symbols numbers = new Symbols(strings[i]);

Tuple<int, int> output = numbers.CheckAndPrint();

Assert.AreEqual(output.Item1, results[i].Item1);

Assert.AreEqual(output.Item2, results[i].Item2);

}

}

[TestMethod]

public void TestNumber2()

{

for (int i = 0; i < strings.Length; i++)

{

Symbols numbers = new Symbols(null);

numbers.Check(strings[i]);

Assert.AreEqual(numbers.countForwardSlash, results[i].Item1);

Assert.AreEqual(numbers.countExclamationMark, results[i].Item2);

}

}

}

}

4 РЕЗУЛЬТАТЫ

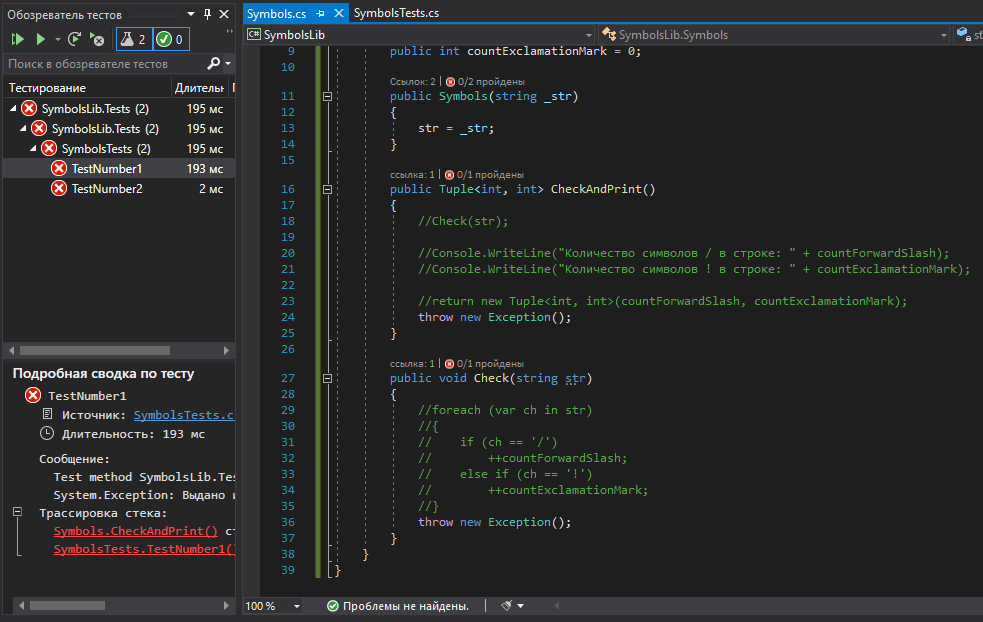


Рисунок 1 – Результаты тестирования до того, как была написана программа

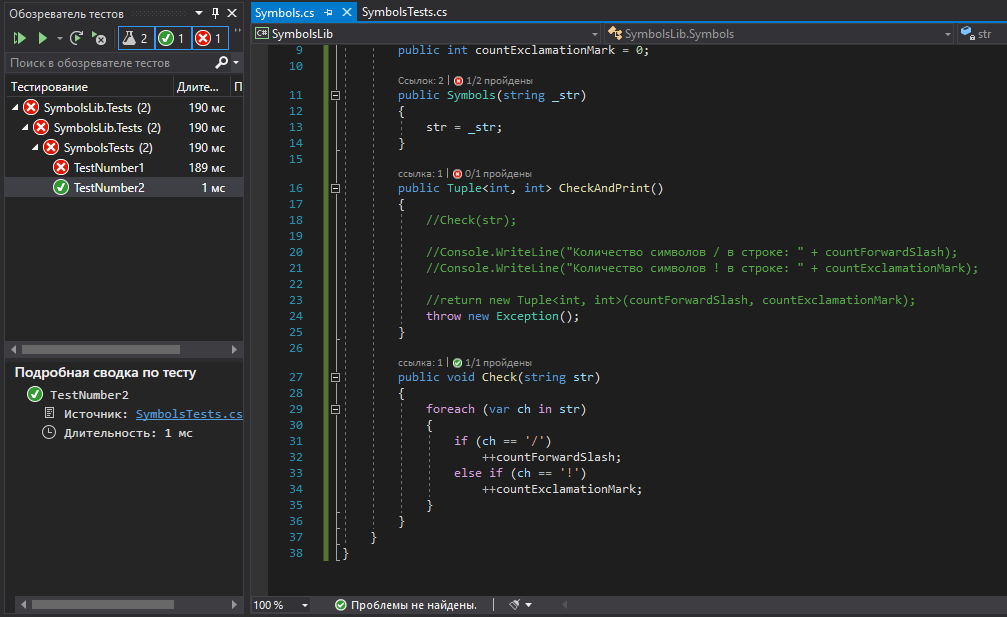


Рисунок 2 – Результаты тестирования с только 1 реализованным методом

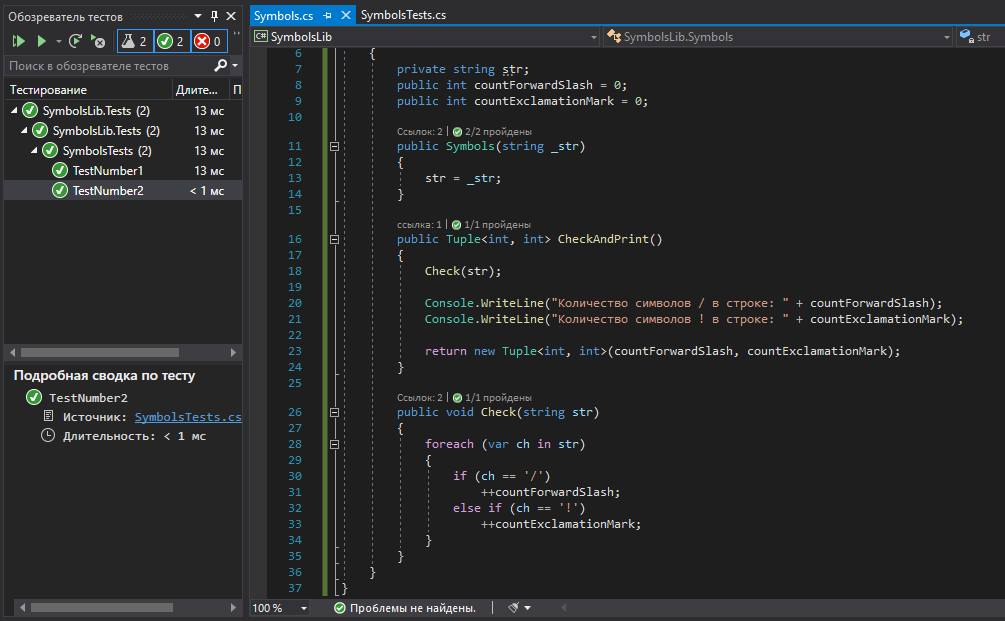


Рисунок 3 – Результаты тестирования полностью рабочей программы

ВЫВОД

В ходе выполнения лабораторной работы были получены практические навыки использования фреймворка NUnit для модульного тестирования программного обеспечения. Был разработан набор тестов, тестирующий каждый метод программы, целью которой является подсчет знаков “/” и “!” в входной строке.