Модульное тестирование в Visual Studio

Модульное тестирование (или Unit-тестирование) предназначено для проверки правильности выполнения небольшого блока кода, решающего свою конкретную задачу. В статье рассказывается, как проводить в модульное тестирование в Visual Studio. Разработка ведётся на языке C#.

Создание проекта программы, модули которой будут тестироваться

Разработаем проект содержащий класс, который вычисляет площадь прямоугольника по длине двух его сторон.

Создадим в Visual Studio новый проект Visual C# -> Библиотека классов. Назовём его **MathTaskClassLibrary**.

Class1 переименуем в **Geometry**.

В классе реализуем метод, вычисляющий площадь прямоугольника. Для демонстрации остановимся на работе с целыми числами. Код программы приведён ниже.

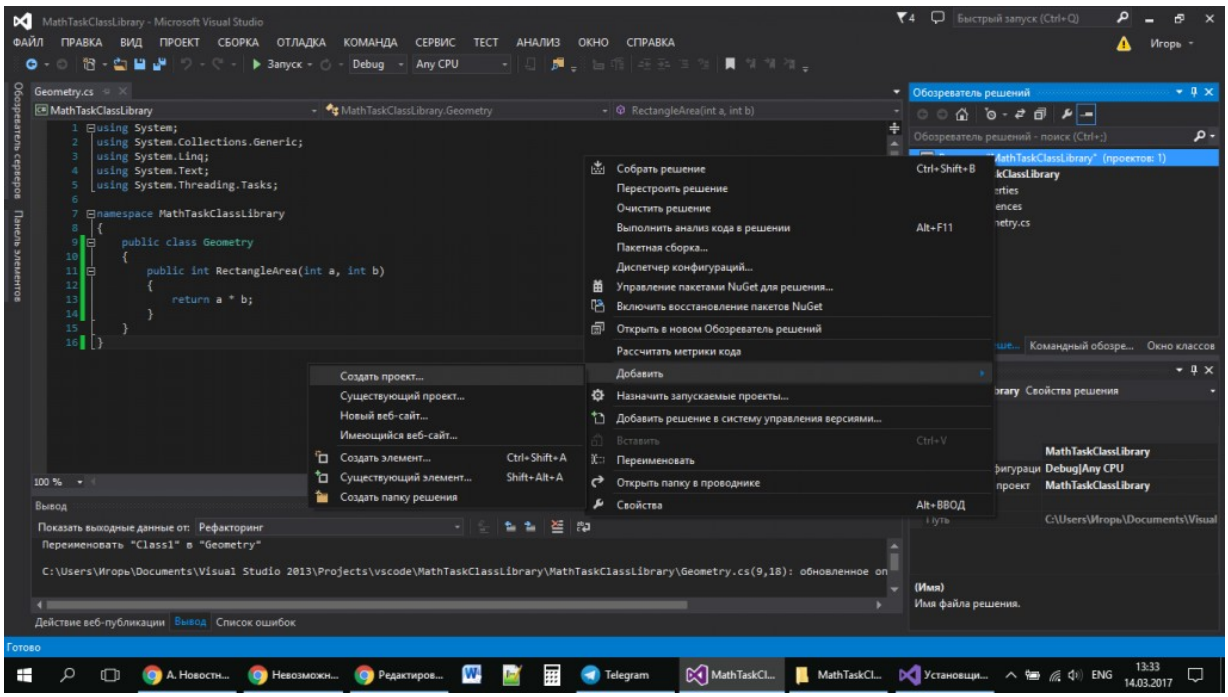
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MathTaskClassLibrary
{
    public class Geometry
    {
        public int RectangleArea(int a, int b)
        {
            return a * b;
        }
    }
}
```

Площадь прямоугольника, как известно, это произведение двух его сторон.

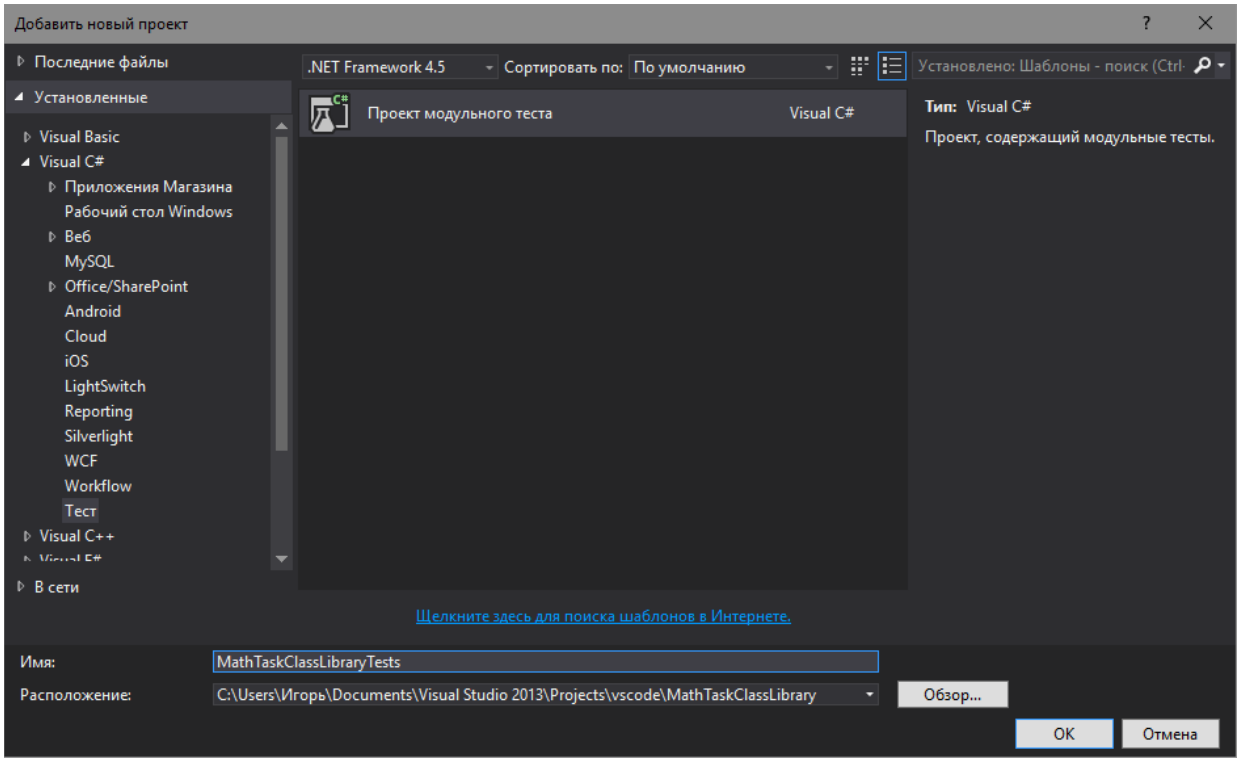
Создание проекта для модульного тестирования в Visual Studio

Чтобы выполнить unit-тестирование, необходимо в рамках того же самого решения создать ещё один проект соответствующего типа.

Правой кнопкой щёлкните по решению, выберите “Добавить” и затем “Создать проект…”.



В открывшемся окне в группе Visual C# щёлкните “Тест”, а затем выберите “Проект модульного теста”. Введите имя проекта **MathTaskClassLibraryTests** и нажмите “OK”. Таким образом проект будет создан.



Перед Вами появится следующий код:

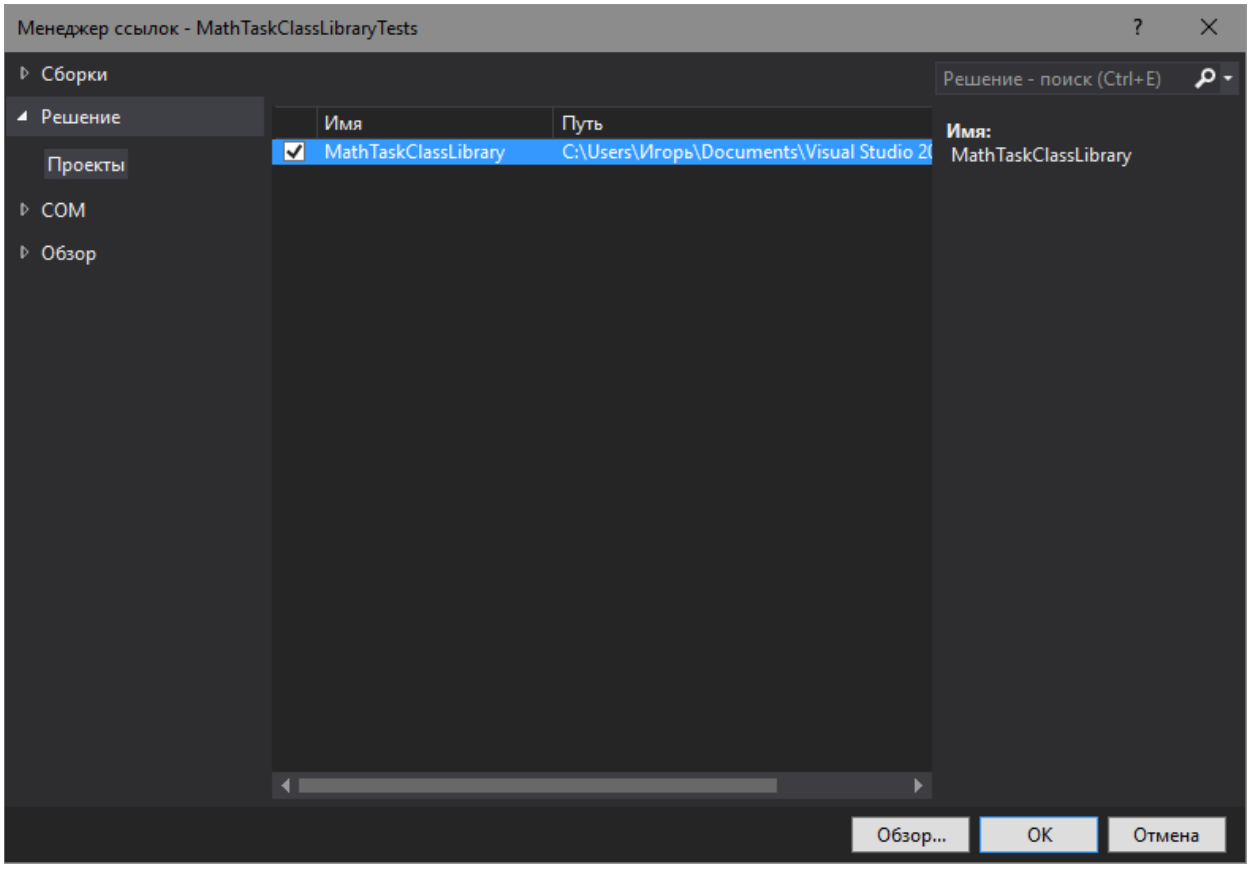
```
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace MathTaskClassLibraryTests
{
    [TestClass]
    public class UnitTest1
    {
        [TestMethod]
        public void TestMethod1()
        {
        }
    }
}
```

Директива [TestMethod] обозначает, что далее идёт метод, содержащий модульный (unit) тест. А [TestClass] в свою очередь говорит о том, что далее идёт класс, содержащий методы, в которых присутствуют unit-тесты.

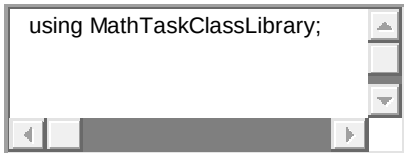
В соответствии с принятыми соглашениями переименуем класс UnitTest1 в **GeometryTests**.

Затем в References проекта необходимо добавить ссылку на проект, код которого будем тестировать. Правой кнопкой щёлкаем на References, а затем выбираем “Добавить ссылку…”.

В появившемся окне раскрываем группу “Решение”, выбираем “Проекты” и ставим галочку напротив проекта **MathTaskClassLibrary**. Затем жмём “OK”.



Также в коде необходимо подключить с помощью директивы using следующее пространство имён:



1 using MathTaskClassLibrary;

Займёмся написание теста. Проверим правильно ли вычисляет программа площадь прямоугольника со сторонами 3 и 5. Ожидаемый результат (правильное решение) в данном случае это число 15.

Переименуем метод TestMethod1() в **RectangleArea_3and5_15returned()**. Новое название метода поясняет, что будет проверяться (RectangleArea – площадь прямоугольника) для каких значений (3 и 5) и что ожидается в качестве правильного результата (15 returned).

Тестирующий метод обычно содержит три необходимых компонента:

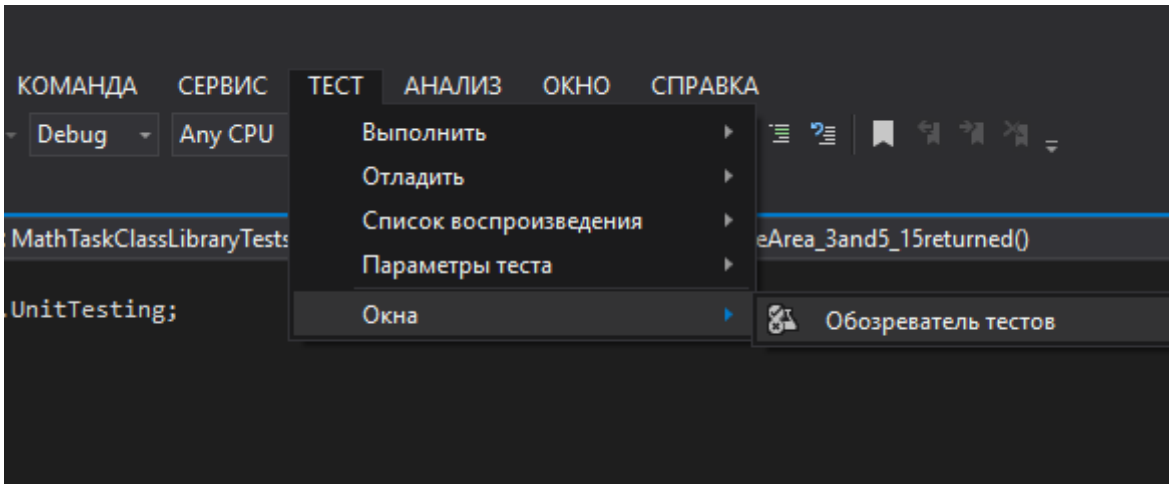
1. исходные данные: входные значения и ожидаемый результат;
2. код, вычисляющий значение с помощью тестируемого метода;
3. код, сравнивающий ожидаемый результат с полученным.

Соответственно тестирующий код будет таким:

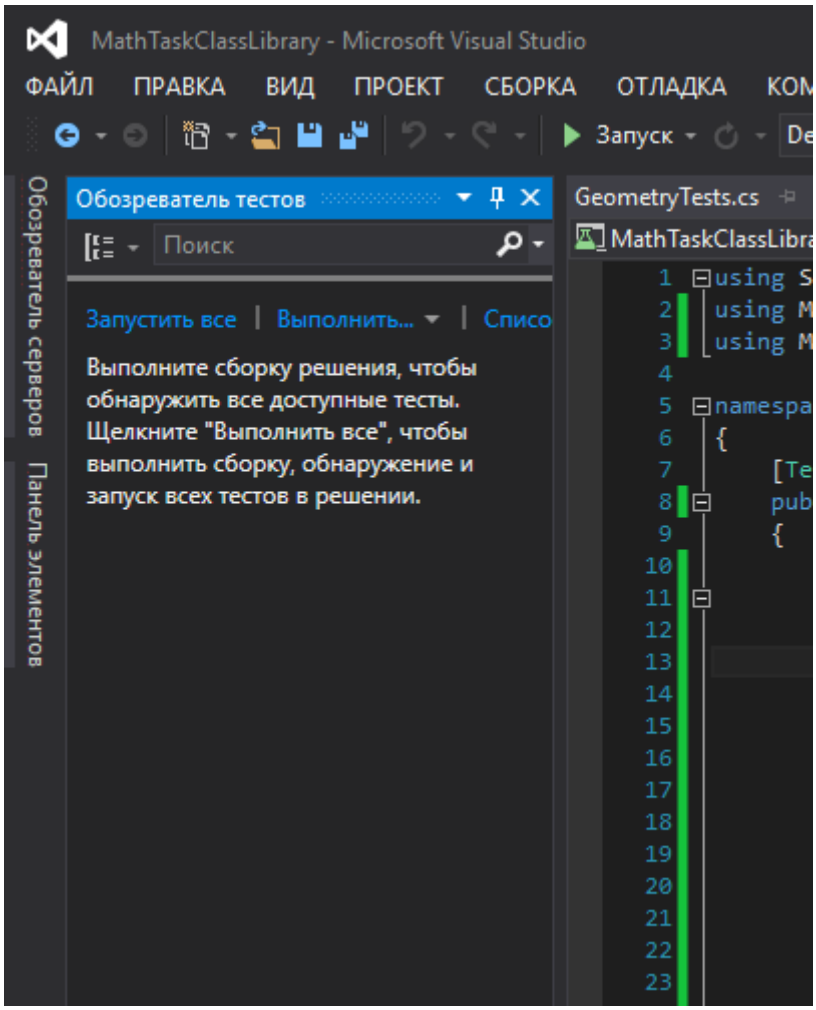
```
1  using System;
2  using Microsoft.VisualStudio.TestTools.UnitTesting;
3  using MathTaskClassLibrary;
4
5  namespace MathTaskClassLibraryTests
6  {
7      [TestClass]
8      public class GeometryTests
9      {
10         [TestMethod]
11         public void RectangleArea_3and5_15returned()
12         {
13             // исходные данные
14             int a = 3;
15             int b = 5;
16             int expected = 15;
17
18             // получение значения с помощью тестируемого метода
19             Geometry g = new Geometry();
20             int actual = g.RectangleArea(a, b);
21
22             // сравнение ожидаемого результата с полученным
23             Assert.AreEqual(expected, actual);
24         }
25     }
26 }
```

Для сравнения ожидаемого результата с полученным используется метод **AreEqual** класса **Assert**. Данный класс всегда используется при написании unit тестов в Visual Studio.

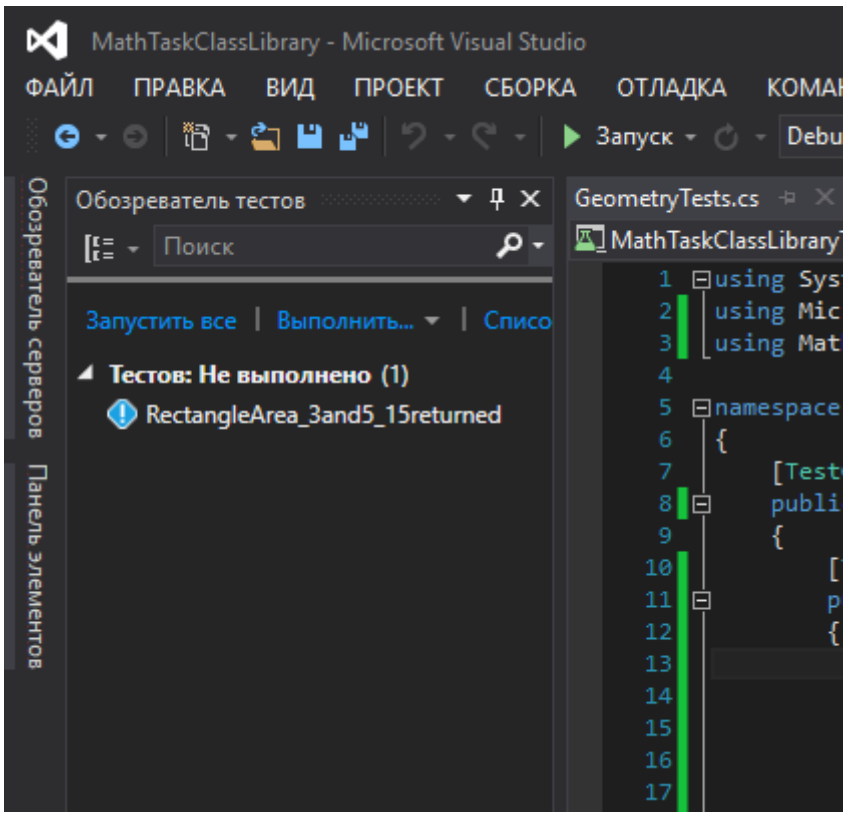
Теперь, чтобы просмотреть все тесты, доступные для выполнения, необходимо открыть окно “Обозреватель тестов”. Для этого в меню Visual Studio щёлкните на кнопку “ТЕСТ”, выберите “Окна”, а затем нажмите на пункт “Обозреватель тестов”.



В студии появится следующее окно:

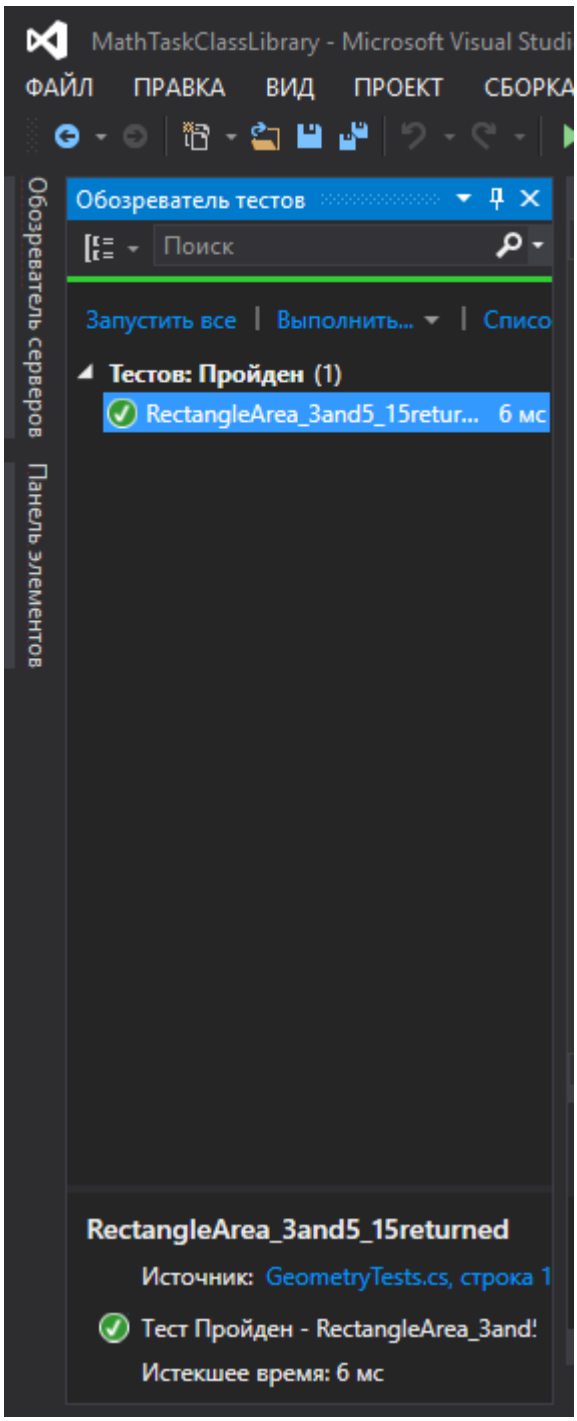


В данный момент список тестов пуст, поскольку решение ещё ни разу не было собрано. Выполним сборку нажатием клавиш Ctrl + Shift + B. После её завершения в “Обозревателе тестов” появится наш тест.



Синяя табличка с восклицательным знаком означает, что указанный тест никогда не выполнялся. Выполним его.

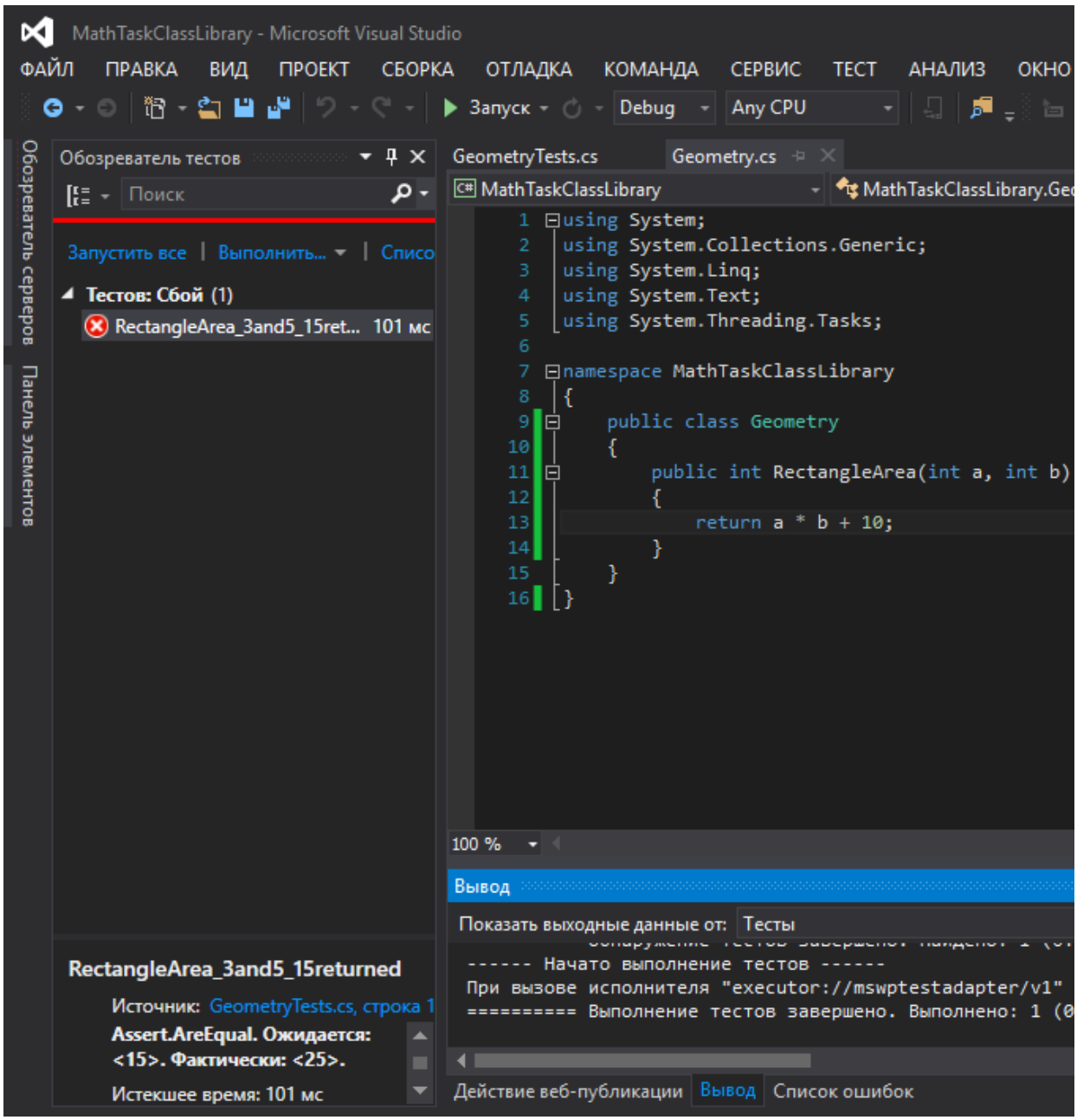
Для этого нажмём правой кнопкой мыши на его имени и выберем “Выполнить выбранные тесты”.



Зелёный кружок с галочкой означает, что модульный тест успешно пройден: ожидаемый и полученный результаты равны.

Изменим код метода **RectangleArea**, вычисляющего площадь прямоугольника, чтобы сымитировать провал теста и посмотреть, как поведёт себя Visual Studio. Прибавим к возвращаемому значению 10.

Запустим unit-тест.



Как Вы видите, красный круг с крестиком показывает провал модульного теста, а ниже указано, что при проверке ожидалось значение 15, а по факту оно равно 25.

Таким образом мы рассмотрели на практике модульное тестирование программы на языке C# в Visual Studio.

Тестирование программного обеспечения – рекомендации

Приведём правило, которым следует руководствоваться при написании и проведении тестов для оценки правильного функционирования программ.

Удобнее всего будет рассмотреть пример основанный на математике.

Так или иначе тестируемый метод или функция (или вся программа в целом) имеет свою область *допустимых* входных значений. **Для проверки правильности работы метода *достаточно* провести тестирование метода на входных значениях начала и конца области допустимых значений (ОДЗ), одного значения из внутренней части области, а также -1 от левой и +1 от правой границы области**.

Например, если ОЗД функции F – это отрезок [0; 100], то для проверки корректности работы функции достаточно протестировать следующие варианты: F(0), F(50) [не обязательно 50, можно взять любое число из внутренней части ОДЗ], F(100), F(-1), F(101).