

# Ну очень первая лекция по



- 1) Microsoft Testing Framework
- 2) Создание тестового проекта, Test Code
- 3) AAA, asserts, SetUp, TearDown
- 4) Запуск тестов
- 5) Источник данных (data driven tests)
- 6) Тестирование private функций
- 7) Плохо тестируемый код

# TDD (Сначала пишем тесты, потом ~ код)

**Задача**

Программа для нахождения корней  
квадратного уравнения  $ax^2 + bx + c = 0$

**Тесты**

$$\begin{array}{ll} x^2 + x - 6 = 0 & \Rightarrow 2, -3 \\ x^2 + 2x + 1 = 0 & \Rightarrow -1, -1 \\ x^2 + 0x - 4 = 0 & \Rightarrow 2, -2 \end{array}$$

(первая итерация)

**Проектирование /  
рефакторинг**

1. Класс `EquationSolver` с методом `Solve()`
2. Подумываем об интерфейсе `IEquationSolver`

**Код**

# TDD

Требование  
уточняется

Надо также проверять явно ситуацию отсутствия  
вещественных корней + порядок корней не важен

Тесты

Корректируем тесты под новые требования +  
сами что-то вспомнили допроверить

$x^2 + x + 1 = 0$	$\Rightarrow$	Exception
$4x^2 = 0$	$\Rightarrow$	0, 0
...		

Проектирование /  
рефакторинг

Корректируем код под новые тесты:

Код

- Вводим exception для ситуации с комплексными корнями
- Меняем сигнатуру метода
- ...

# TDD

**Требование  
уточняется**

Нужно также решать уравнения больших порядков

**Тесты**

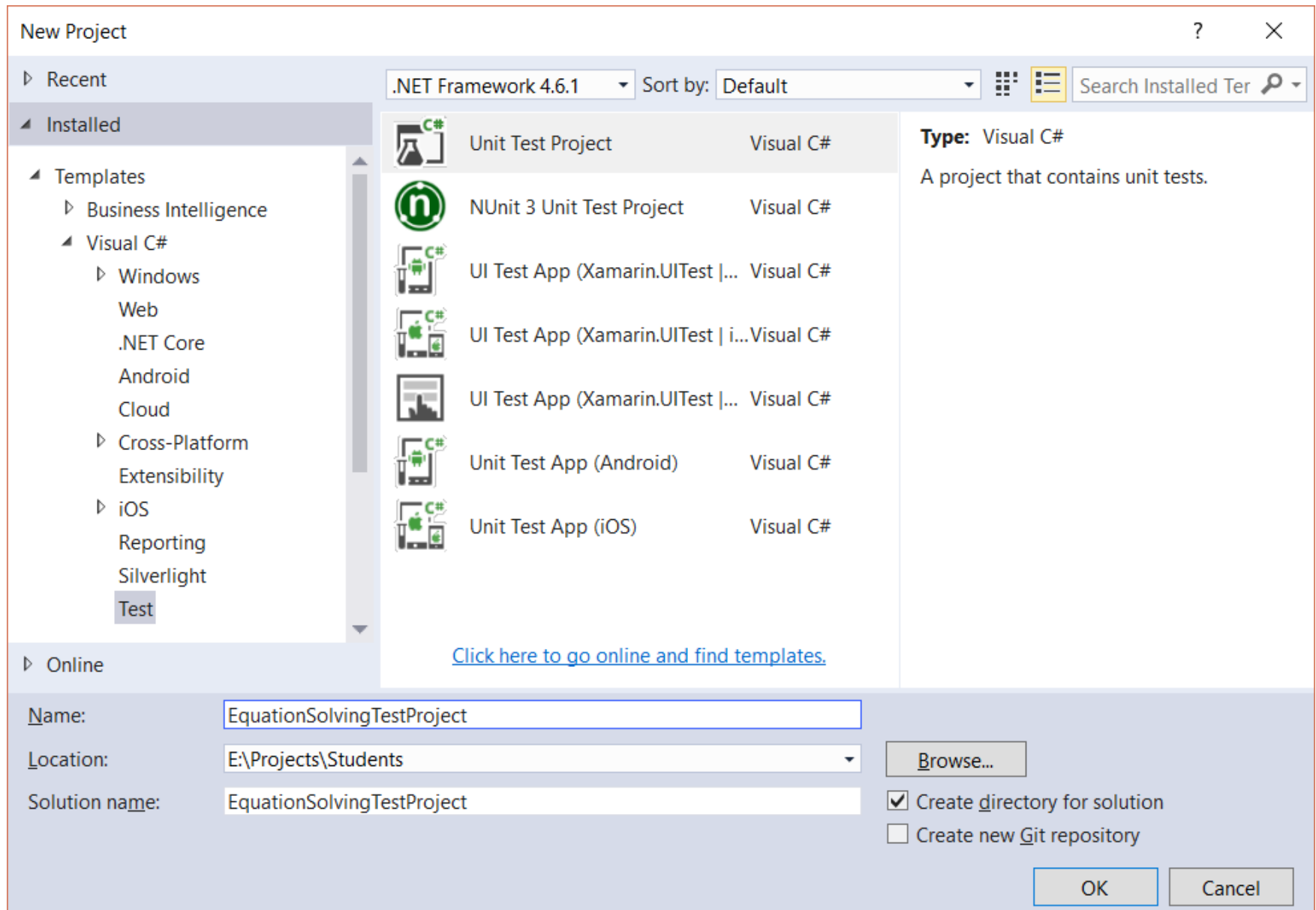
Корректируем тесты под новые требования

**Проектирование /  
рефакторинг**

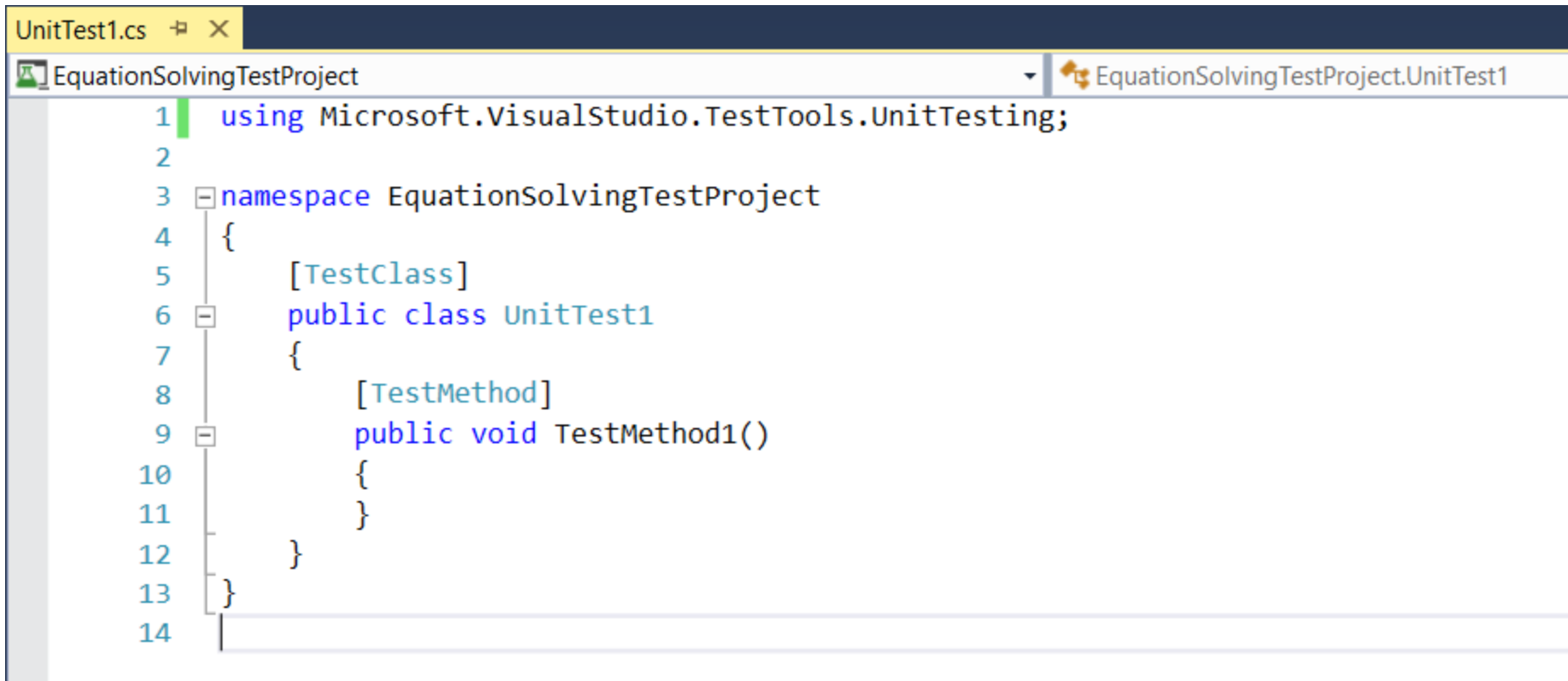
Корректируем код под новые тесты

**Код**

# Создаем в VisualStudio проект Test Project



# Создаем в VisualStudio проект Test Project



The screenshot shows the Visual Studio code editor with a file named `UnitTest1.cs` open. The project is `EquationSolvingTestProject`, and the current file is `EquationSolvingTestProject.UnitTest1`. The code is written in C# and defines a test class `UnitTest1` within the `EquationSolvingTestProject` namespace. The code includes the `using Microsoft.VisualStudio.TestTools.UnitTesting;` directive, a `[TestClass]` attribute on the `UnitTest1` class, and a `[TestMethod]` attribute on the `TestMethod1()` method. The method body is currently empty.

```
1 using Microsoft.VisualStudio.TestTools.UnitTesting;
2
3 namespace EquationSolvingTestProject
4 {
5     [TestClass]
6     public class UnitTest1
7     {
8         [TestMethod]
9         public void TestMethod1()
10        {
11        }
12    }
13 }
14
```

# Пишем код теста

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

```
namespace EquationSolvingTestProject
{
    [TestClass]
    public class QuadraticEquationUnitTest
    {
        [TestMethod]
        public void TestTwoDifferentRoots()
        {
            EquationSolver solver = new EquationSolver();

            double[] roots = solver.Solve(1, 1, -6);

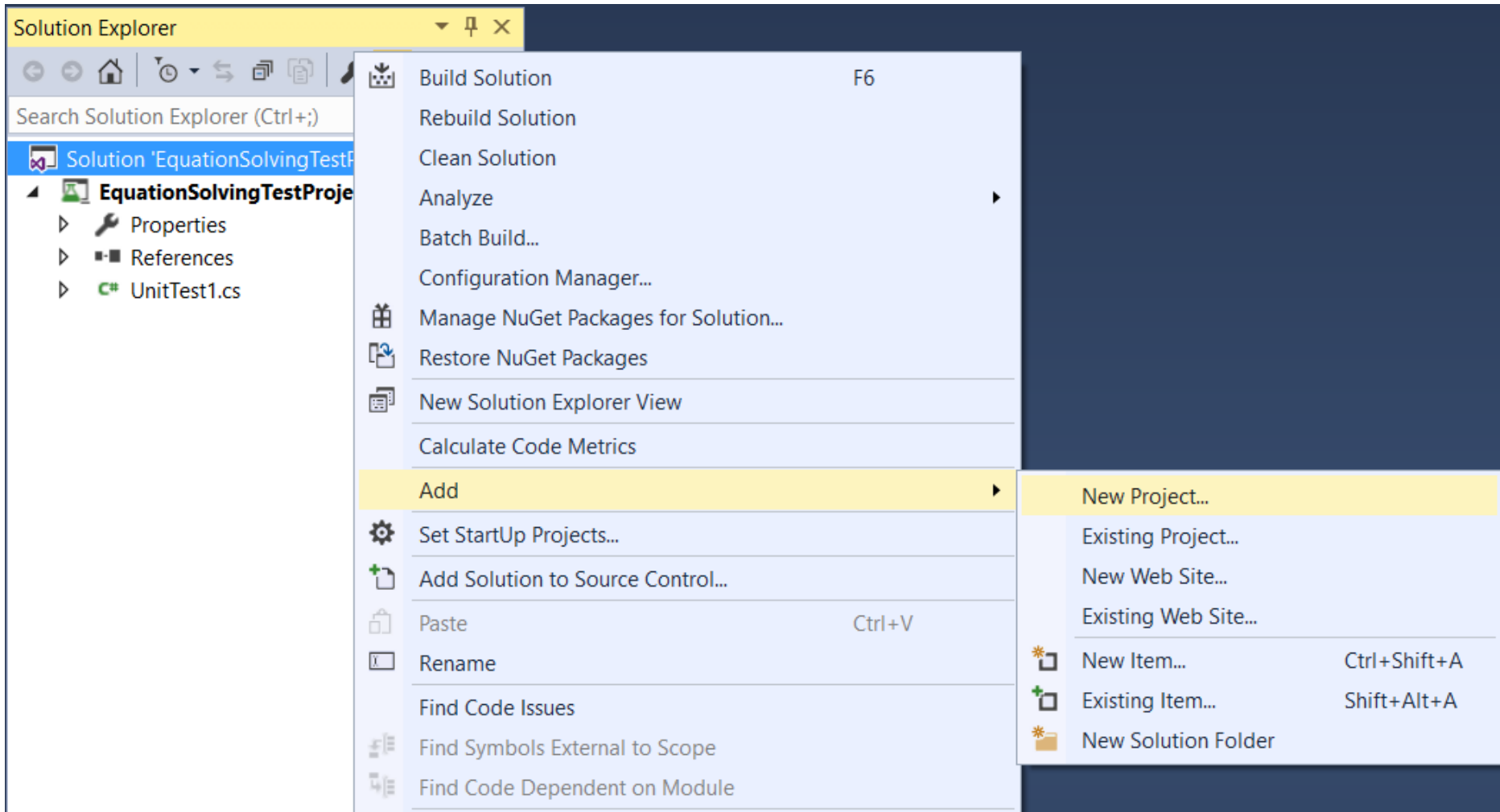
            Assert.AreEqual(roots[0], 2);
            Assert.AreEqual(roots[1], -3);
        }

        [TestMethod]
        public void TestOneRoot()
        {
            // ...
        }

        [TestMethod]
        public void TestZeroCoefficients()
        {
            // ...
        }
    }
}
```

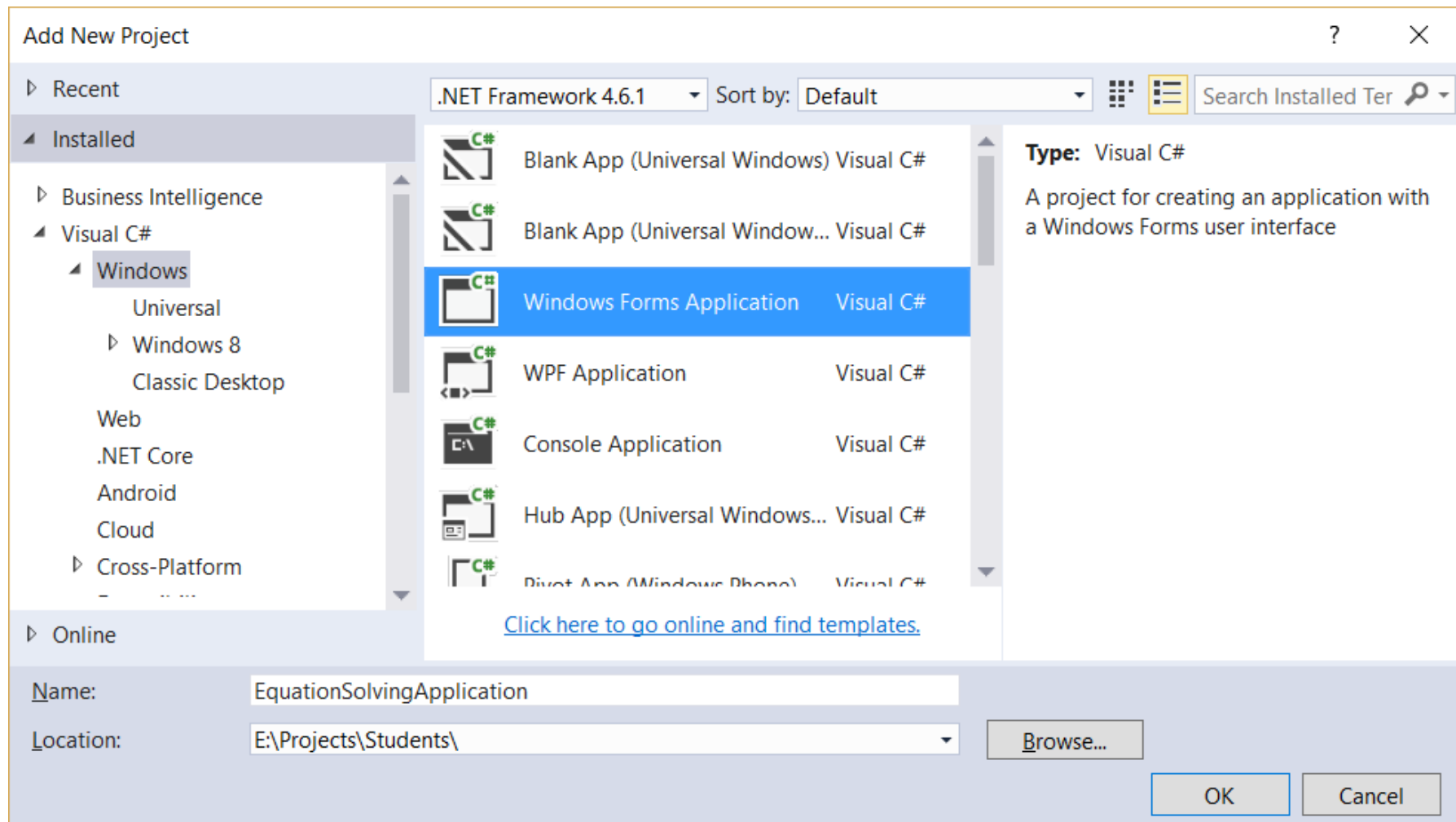
**Этот тест будет проверять корректность работы класса-решателя алгебраических уравнений EquationSolver (сам класс пока мы еще не написали, поэтому Студия негодует)**

# Пишем production code

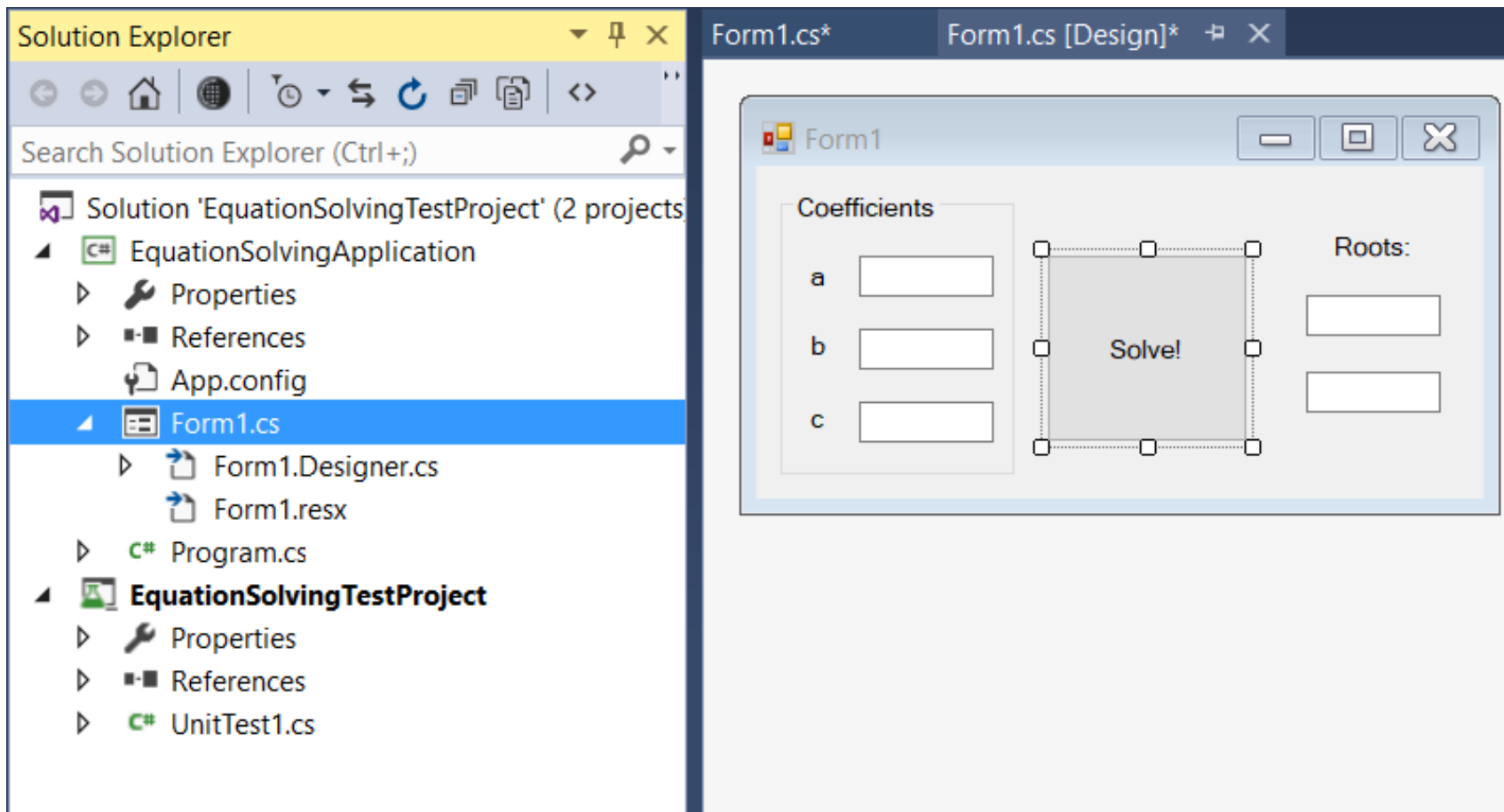




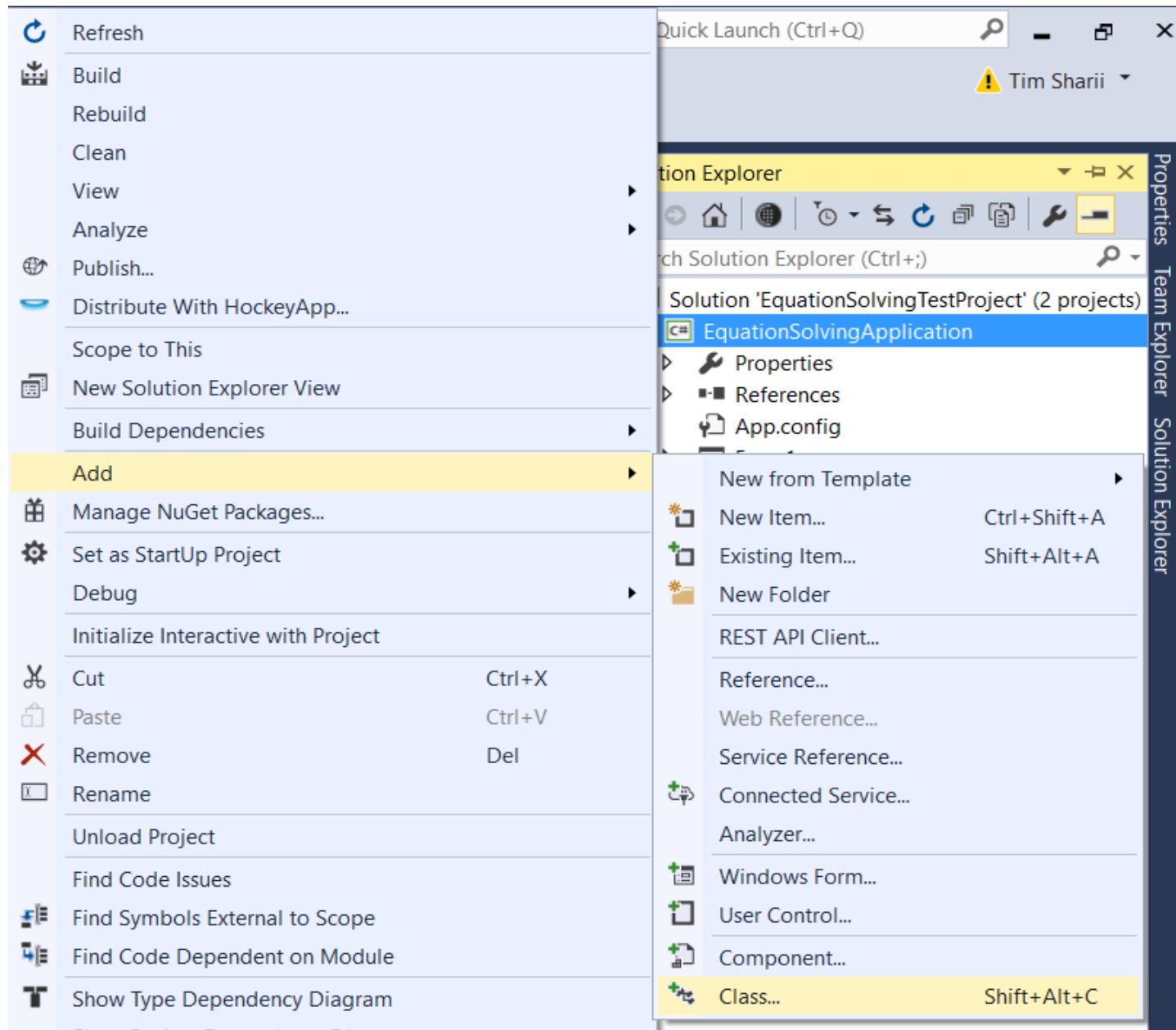
# Создаем проект самого приложения



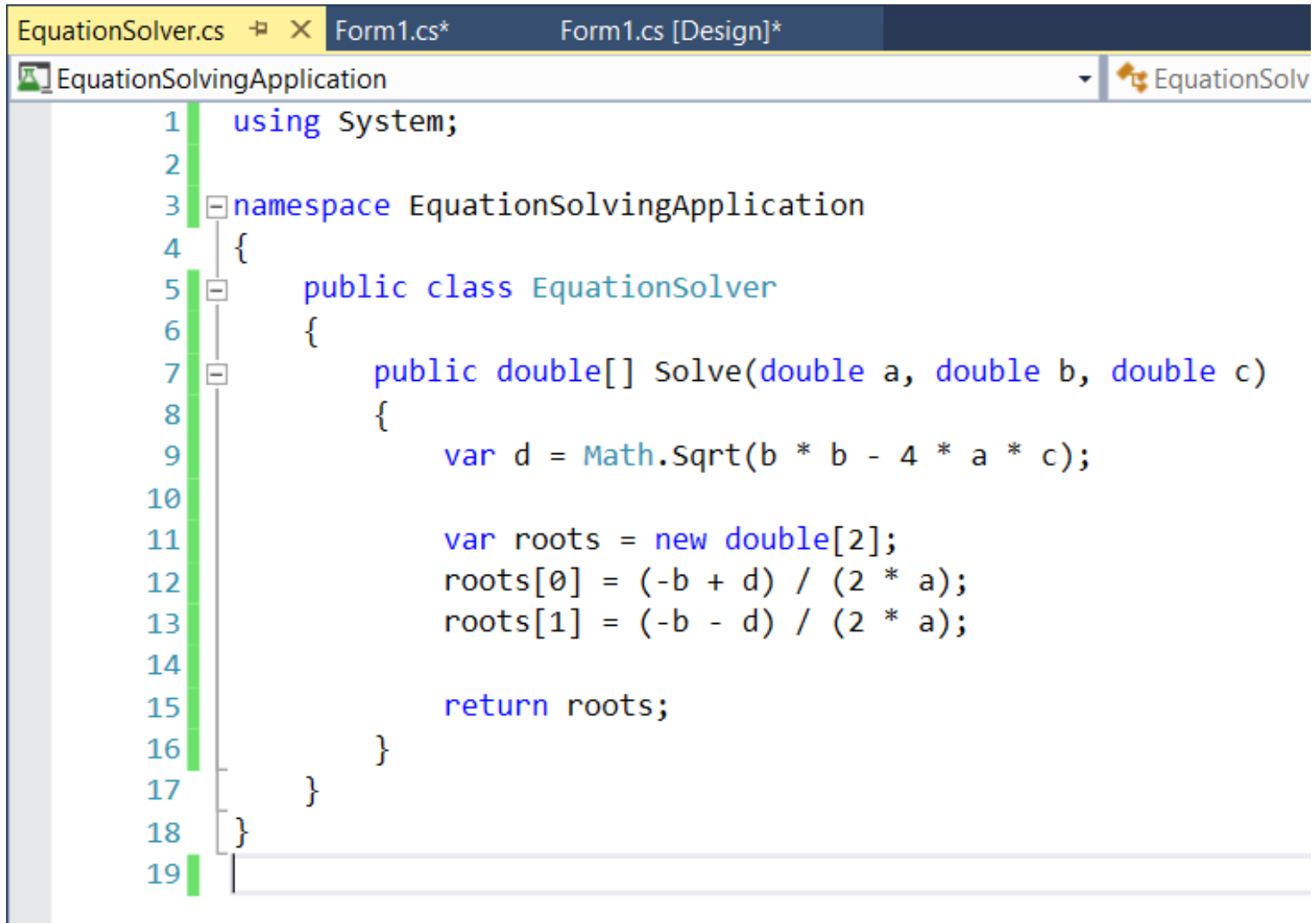
# Создаем проект самого приложения



# Добавляем класс EquationSolver



# Пишем код класса



```
EquationSolver.cs  Form1.cs*  Form1.cs [Design]*
EquationSolvingApplication
1  using System;
2
3  namespace EquationSolvingApplication
4  {
5      public class EquationSolver
6      {
7          public double[] Solve(double a, double b, double c)
8          {
9              var d = Math.Sqrt(b * b - 4 * a * c);
10
11              var roots = new double[2];
12              roots[0] = (-b + d) / (2 * a);
13              roots[1] = (-b - d) / (2 * a);
14
15              return roots;
16          }
17      }
18  }
```

# Напишем сразу и немного кода окна (хотя тест прогнать уже можно и без этого)

```
cs Form1.cs Form1.cs [Design]
using Application
using System;
using System.Windows.Forms;

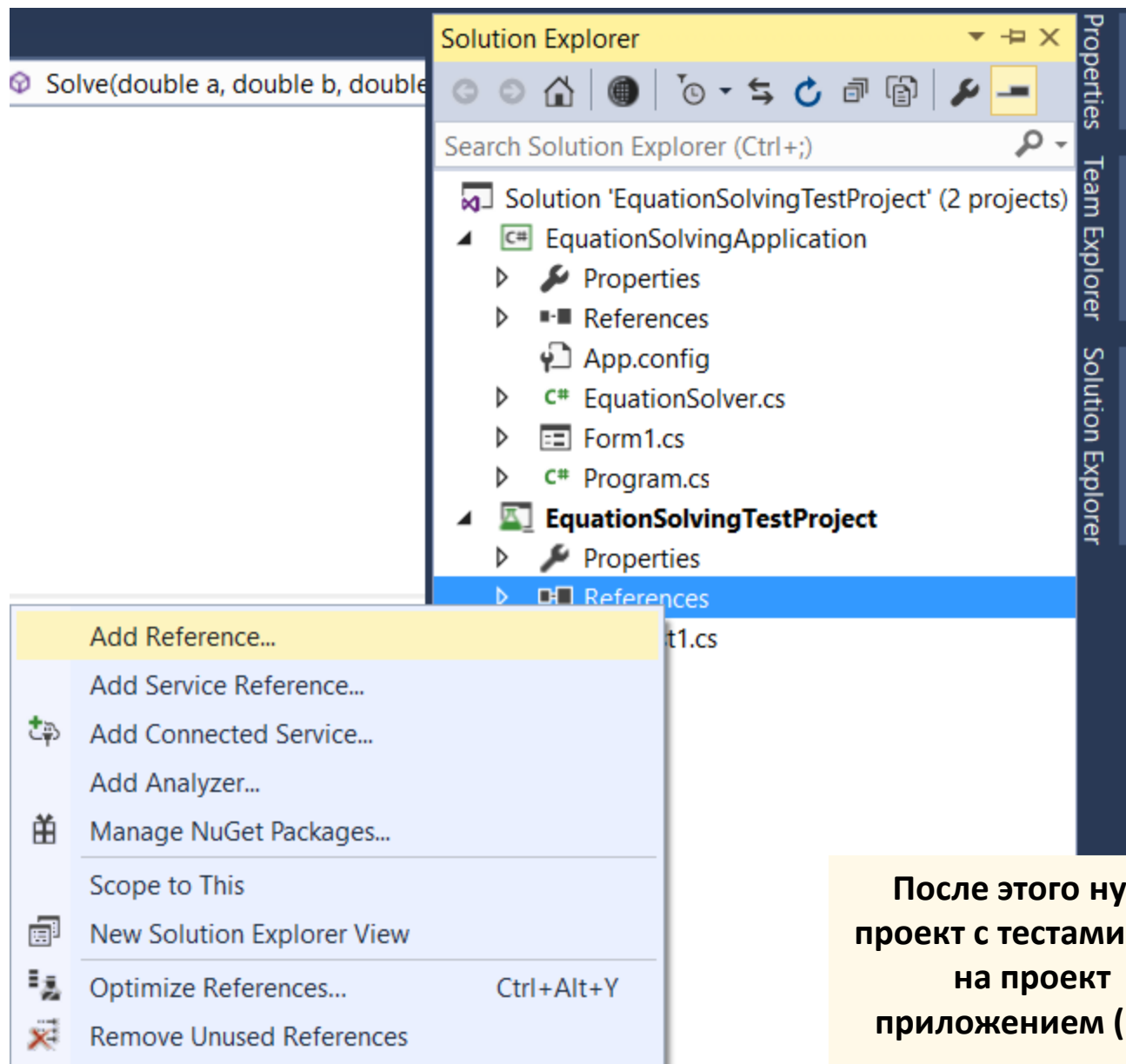
namespace EquationSolvingApplication
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void buttonSolve_Click(object sender, EventArgs e)
        {
            var solver = new EquationSolver();

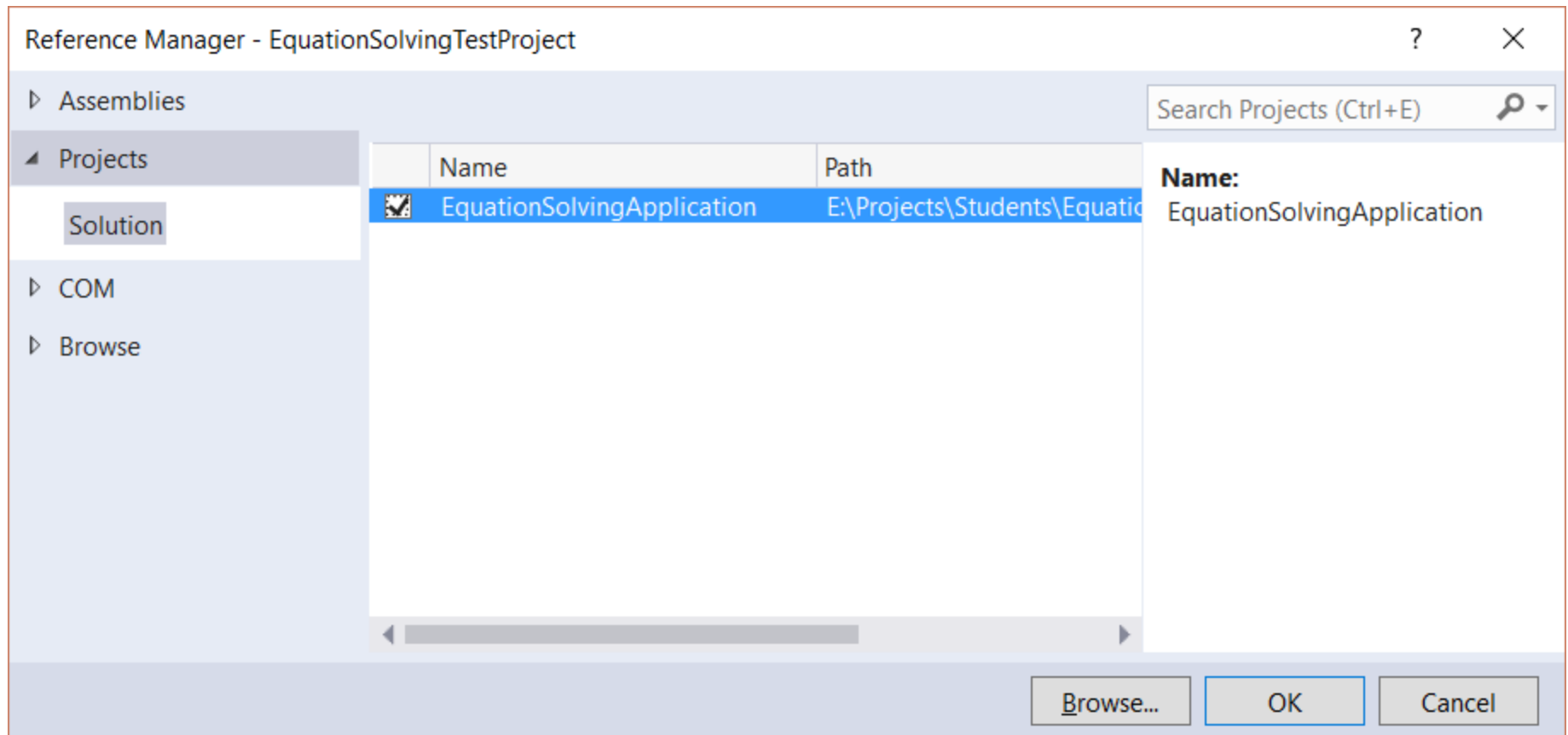
            var roots = solver.Solve(double.Parse(textBoxA.Text),
                                     double.Parse(textBoxB.Text),
                                     double.Parse(textBoxC.Text));

            textBoxRoot1.Text = roots[0].ToString("F3");
            textBoxRoot2.Text = roots[1].ToString("F3");
        }
    }
}
```

**Запустим основное приложение.  
Оно должно нормально отработать.**



**После этого нужно добавить в проект с тестами (test code) ссылку на проект с основным приложением (production code).**



```

using EquationSolvingApplication;
using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace EquationSolvingTestProject
{
    [TestClass]
    public class QuadraticEquationUnitTest
    {
        [TestMethod]
        public void TestTwoDifferentRoots()
        {
            EquationSolver solver = new EquationSolver();

            double[] roots = solver.Solve(1, 1, -6);

            Assert.AreEqual(roots[0], 2);
            Assert.AreEqual(roots[1], -3);
        }

        [TestMethod]
        public void TestOneRoot()
        {
            // ...
        }

        [TestMethod]
        public void TestZeroCoefficients()
        {
            // ...
        }
    }
}

```



**Теперь проект с тестами должен нормально компилироваться (указать неймспейс :  
using EquationSolvingApplication).**



```

using EquationSolvingApplication;
using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace EquationSolvingTestProject
{
    [TestClass]
    public class QuadraticEquationUnitTests
    {
        [TestMethod]
        public void TestTwoDifferentRoots()
        {
            EquationSolver solver = new EquationSolver();

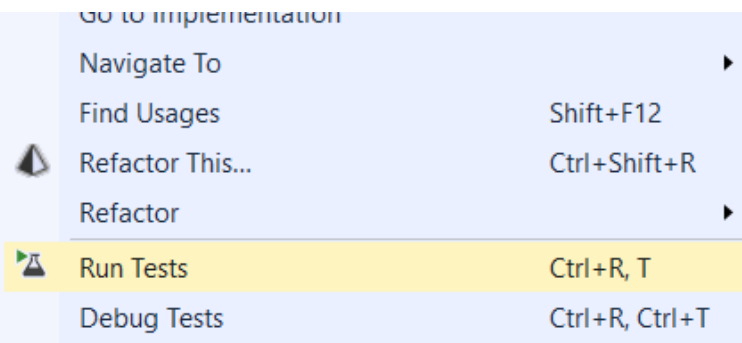
            double[] roots = solver.Solve(1, -3, 2);

            Assert.AreEqual(roots[0], 2);
            Assert.AreEqual(roots[1], -3);
        }

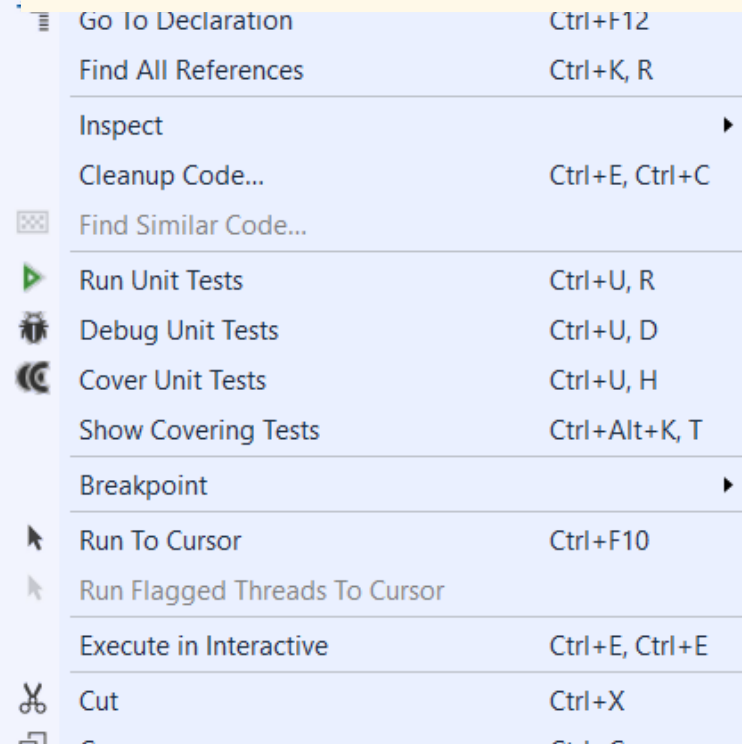
        [TestMethod]
        public void TestOneRoot()
        {
            // ...
        }

        [TestMethod]
        public void TestZeroCoefficients()
        {
            // ...
        }
    }
}

```



**Для запуска тестов –  
правой кнопкой мыши меню + Run Tests**



Toolbox

Server Explorer

Test Explorer

Test Explorer

Search

Configure continuous integration

Setup continuous integration(CI) builds to test continuously after every code change.

Don't show this again

Run All

|

Run...

|

Playlist : All Tests

Passed Tests (3)

✓

TestOneRoot

< 1 ms

✓

TestTwoDifferentRoots

5 ms

✓

TestZeroCoefficients

< 1 ms

Form1.cs

Form1.cs [Design]

Equation

ingApplication;

VisualStudio.TestTools.UnitTesting;

SolvingTestProject

QuadraticEquationUnitTest

d]

d TestTwoDifferentRoots()

onSolver solver = new EquationSolver();

[] roots = solver.Solve(1, 1, -6);

.AreEqual(roots[0], 2);

.AreEqual(roots[1], -3);

## Принцип организации теста AAA:

A – Arrange

A – Act

A – Assert (самое главное в тесте, проверка)

```
[TestClass]
public class QuadraticEquationUnitTest
{
    [TestMethod]
    public void TestTwoDifferentRoots()    // "AAA" : Triple A
    {
        // ARRANGE
        EquationSolver solver = new EquationSolver();

        // ACT
        double[] roots = solver.Solve(1, 1, -6);

        // ASSERT
        Assert.AreEqual(roots[0], 2);
        Assert.AreEqual(roots[1], -3);
    }
}
```

```

#region Additional test attributes
//
//You can use the following additional attributes as you write your tests:
//
//Use ClassInitialize to run code before running the first test in the class
//[ClassInitialize()]
setUp //public static void MyClassInitialize(TestContext testContext)
//{
//}
//
//Use ClassCleanup to run code after all tests in a class have run
//[ClassCleanup()]
tearDown //public static void MyClassCleanup()
//{
//}
//
//Use TestInitialize to run code before running each test
//[TestInitialize()]
//public void MyTestInitialize()
//{
//}
//
//Use TestCleanup to run code after each test has run
//[TestCleanup()]
//public void MyTestCleanup()
//{
//}
//
#endregion

```

```

[TestClass]
public class QuadraticEquationUnitTest
{
    readonly EquationSolver _solver = new EquationSolver();

    [TestInitialize]
    public void QuadraticEquationTestInitialize()
    {
        // additional test setup logic
    }

    [TestMethod]
    public void TestTwoDifferentRoots()    // "AAA" : Triple A
    {
        // ACT
        double[] roots = _solver.Solve(1, 1, -6);

        // ASSERT
        Assert.AreEqual(roots[0], 2);
        Assert.AreEqual(roots[1], -3);
    }

    [TestMethod]
    public void TestOneRoot()
    {
        // ACT
        double[] roots = _solver.Solve(1, 2, 1);

        // ASSERT
        Assert.AreEqual(roots[0], -1);
    }
}

```

**Т.к. solver потокобезопасный и не имеет внутреннего состояния, то он может быть разделен между тестами**

## Пример «зафейлившегося» теста

(мы в продакшне вообще не предусмотрели этот случай, а если бы этот тест был написан сразу по TDD, то не пропустили бы эту спецификацию)

```
[TestMethod]
public void TestZeroCoefficients()
{
    double[] roots = _solver.Solve(0, 1, 2);

    // ASSERT
    Assert.AreEqual(roots[0], -2);
}
```

The screenshot shows the Test Explorer window in Visual Studio. At the top, there's a search bar and a red bar with the text "Configure continuous integration" and "Setup continuous integration(CI) builds to test continuously after every code change." Below this, there are buttons for "Run All", "Run...", and "Playlist : All Tests". The main section is titled "Failed Tests (1)" and shows a single failed test: "TestZeroCoefficients" with a red 'X' icon and a duration of 7 ms. Below this, there's a section for "Passed Tests (2)" showing "TestOneRoot" (3 ms) and "TestTwoDifferentRoots" (< 1 ms), both with green checkmark icons. The bottom section is titled "TestZeroCoefficients" and shows the source as "UnitTest1.cs line 40". It contains a red 'X' icon and the text "Test Failed - TestZeroCoefficients". The message says "Message: Assert.AreEqual failed. Expected:<не число>. Actual:<-2>." The elapsed time is 7 ms. The stack trace shows "QuadraticEquationUnitTest.TestZer".

Test Explorer

Search

Configure continuous integration  
Setup continuous integration(CI) builds to test continuously after every code change.  
Don't show this again

Run All | Run... | Playlist : All Tests

Failed Tests (1)

TestZeroCoefficients 7 ms

Passed Tests (2)

TestOneRoot 3 ms

TestTwoDifferentRoots < 1 ms

TestZeroCoefficients

Source: UnitTest1.cs line 40

Test Failed - TestZeroCoefficients

Message: Assert.AreEqual failed.  
Expected:<не число>.  
Actual:<-2>.

Elapsed time: 7 ms

StackTrace:  
QuadraticEquationUnitTest.TestZer

На следующей итерации еще подправили код:  
если дискриминант меньше 0 (нет вещественных корней), то бросаем исключение;  
если первые два коэффициента равны 0, то в уравнении нет неизвестных, - то же самое

```
public class EquationSolver
{
    public double[] Solve(double a, double b, double c)
    {
        if (Math.Abs(a) < 1e-10)    // if (a == 0.0)
        {
            if (Math.Abs(b) < 1e-10)
            {
                throw new ArgumentException("No unknowns!");
            }

            return new[] { -c / b };
        }

        var d = b * b - 4 * a * c;

        if (d < 0)
        {
            throw new Exception("No real roots!");
        }

        d = Math.Sqrt(d);

        var roots = new double[2];
        roots[0] = (-b + d) / (2 * a);
        roots[1] = (-b - d) / (2 * a);

        return roots;
    }
}
```

Для коллекций (в т.ч. массивов) можем использовать специальный класс `CollectionAssert`

```
[TestMethod]
public void TestTwoDifferentRoots()    // "AAA" : Triple A
{
    // ACT
    double[] roots = _solver.Solve(1, 1, -6);

    // ASSERT
    CollectionAssert.AreEqual(roots, new [] { 2.0, -3.0 });

    // we could also test:

    // 1) CollectionAssert.AllItemsAreNotNull(roots);
    // 2) CollectionAssert.AllItemsAreUnique(roots);
}
```



А эти тесты не проверяют конкретные числа или коллекции – их задача проверить, что функция действительно *бросает исключение* в ситуации, когда нет вещественных корней уравнения, и в ситуации, когда в уравнении нет неизвестных

```
[TestMethod]
[ExpectedException(typeof(Exception))]
public void TestNoRealRoots()
{
    // ARRANGE, ACT, ASSERT in one line
    _solver.Solve(1, 1, 1);
}


[TestMethod]
[ExpectedException(typeof(ArgumentException))]
public void TestNoUnknowns()
{
    // ARRANGE, ACT, ASSERT in one line
    _solver.Solve(0, 0, 1);
}
```

	A	B	C	D	E	F
1	a	b	c	x1	x2	
2		1	1	-6	2	-3
3		1	2	1	-1	-1
4		1	4	4	-2	-2
5		1	5	4	-1	-4
6		1	6	9	-3	2
7						






На следующем слайде будет код, который позволит вытянуть данные для тестов из источника данных (например, excel-файла)

TestExcel [Results] X
QuadraticEquationSolver.cs
Form1.cs
UnitTest1.cs

Common Results

Test Run: Timothy@ARISTOCRATIC 2014-10-27 19:09:14  
Test Name: TestExcel  
Result:  Failed  
Duration: 00:00:01.2301429  
Computer Name: ARISTOCRATIC  
Start Time: 27.10.2014 19:09:18  
End Time: 27.10.2014 19:09:20

Data Driven Test Results: 4 of 5 passed

Result	Duration	Data Row	Error Message
 Passed	00:00:00.0351475	0	
 Passed	00:00:00.0000597	1	
 Passed	00:00:00.0000175	2	
 Passed	00:00:00.0000163	3	
 Failed	00:00:00.1281571	4	Assert.AreEqual failed. Expected: <-3>. Actual: <2>.

```

[TestClass]
public class QuadraticEquationUnitTestDataSheet
{
    EquationSolver _solver = new EquationSolver();

    const string dataDriver = "System.Data.OleDb";
    const string connectionStr = "Dsn=Excel Files;dbq=D:\\testData.xlsx";

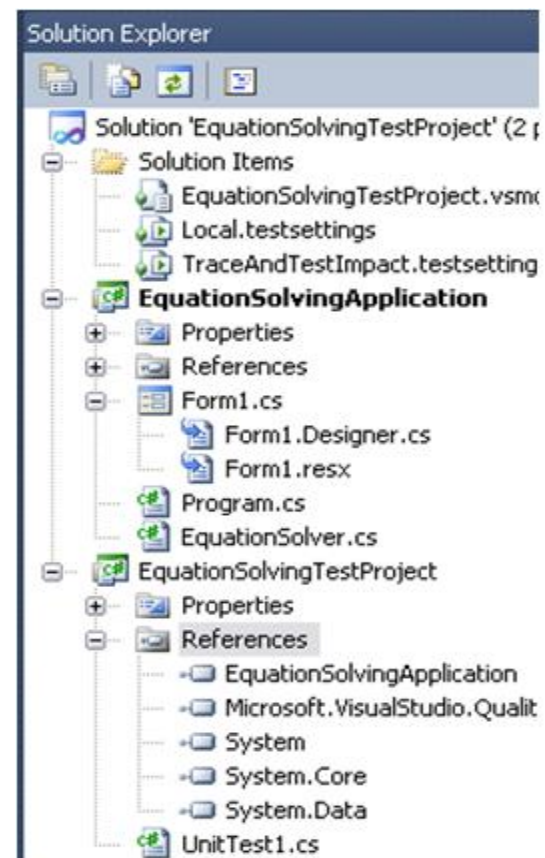
    [TestMethod]
    [DataSource("System.Data.Odbc", connectionStr, "Sheet1$", DataAccessMethod.Sequential)]
    public void TestExcel()
    {
        // ARRANGE
        double a = Convert.ToDouble( TestContext.DataRow["a"] );
        double b = Convert.ToDouble( TestContext.DataRow["b"] );
        double c = Convert.ToDouble( TestContext.DataRow["c"] );
        double r1 = Convert.ToDouble( TestContext.DataRow["x1"] );
        double r2 = Convert.ToDouble( TestContext.DataRow["x2"] );

        // ACT
        double[] roots = _solver.Solve(a, b, c);

        // ASSERT
        Assert.AreEqual( roots[0], r1 );
        Assert.AreEqual( roots[1], r2 );
    }

    public TestContext TestContext { get; set; }
}

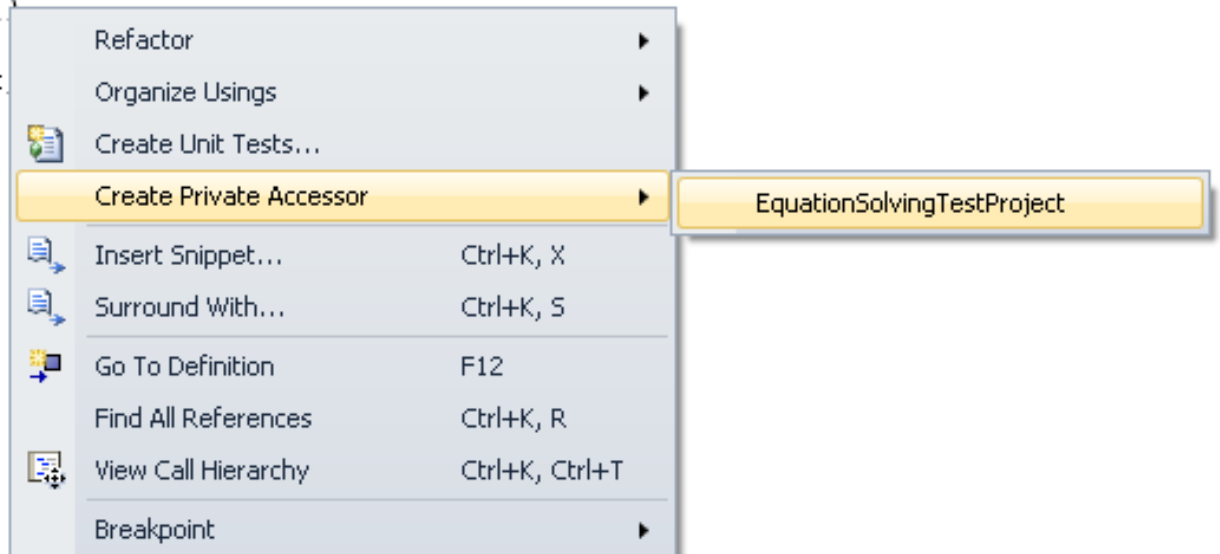
```



Для демки сделаем приватный метод `double Discriminant()`.  
Этот метод будет считать дискриминант на основе коэффициентов `a`, `b`, `c`,  
которые теперь будут задаваться в сеттере объекта `EquationSolver`

Как можно написать юнит-тест для `private` функций:  
Способ 1. Создать т.н. `Accessor`

```
private double Discriminant()  
{  
    return b * b - 4 * a * c;  
}
```



Как можно написать юнит-тест для private функций:  
Способ 1. Создать т.н. Accessor

```
[TestClass]
public class QuadraticEquationUnitTest
{
    EquationSolver _solver = new EquationSolver();

    EquationSolver_Accessor _target = new EquationSolver_Accessor();

    [TestMethod]
    public void TestDiscriminant()
    {
        // ARRANGE
        _target.setCoeffs(1, 1, -6);

        // ACT
        double discr = _target.Discriminant();

        // ASSERT
        Assert.AreEqual( discr, 25 );
    }
}
```

Как можно написать юнит-тест для private функций:  
Способ 2. Обратиться к PrivateObject

```
[TestMethod]
public void TestDiscriminantPrivateObject()
{
    // ARRANGE
    _solver.setCoeffs(1, 1, -6);

    PrivateObject p = new PrivateObject( _solver );

    // ACT
    double discr = (double)p.Invoke( "Discriminant" );

    // ASSERT
    Assert.AreEqual(discr, 25);
}
```