

**TDD with VS**

The purpose of this assignment is to experience TDD using unit testing tools in VS. You are given a walkthrough of using unit tests to develop a method of a class. Also you are given an article that takes issue with TDD and you are required to discuss the it in the group and write conclusions -- it's not so much about a right/wrong answer than an intelligent discussion of merits/demerits of TDD as per your point of view. (If you don't agree for any or all particulars as a group, you are encouraged to say so).

Due

Complete this assignment in class in groups of two at most. Answer the questions on the last page. Sign last page and hand it in before you leave the lab session. Upload the class work in eCentennial.

Marks

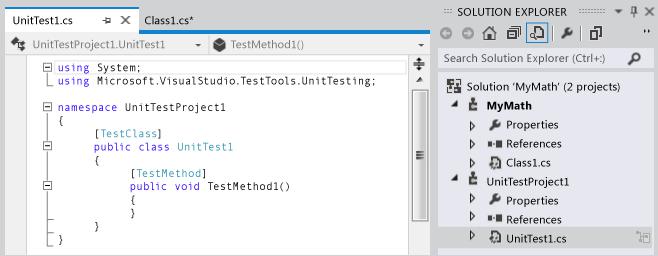
The assignment is marked out of 10.

[**Walkthrough: Using Unit Tests to Develop a Method**](javascript:void(0))

This walkthrough demonstrates how to develop a tested method in C# using the Microsoft Unit Test framework. You can easily adapt it for other languages, and to use other test frameworks such as NUnit. For more information, see [Using Different Test Frameworks.](https://msdn.microsoft.com/en-us/library/hh212233(v=vs.110).aspx#frameworks)

**Creating the Test and Method**

1. Create a Visual C# Class Library project. This project will contain the code that we want to deliver. In this example, it is named **MyMath**.
2. Create a Test project.
3. In the **New Project** dialog, choose **Visual C#**, **Test** and then choose **Unit Test Project**.



1. Write a basic test method. Verify the result obtained for a specific input: C#

[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);

}

1. Generate the method from the test.
   1. Place the cursor on Rooter, and then on the shortcut menu choose **Generate**, **New Type**.
   2. In the **Generate New Type** dialog box, set **Project** to the class library project. In this example, it is **MyMath**.
   3. Place the cursor on SquareRoot, and then on the shortcut menu choose **Generate**, **Method Stub**.
2. On the **Test** menu, choose **Run Unit Tests**, **All Tests**. The solution builds and runs.

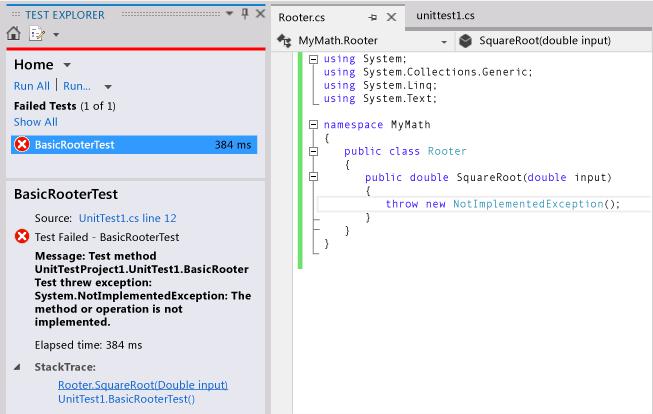
Test Explorer opens and displays the results.

The test appears under **Failed Tests**.

Select the name of the test.

The details of the test appear in the lower part of Test Explorer.

Select the items under **Stack Trace** to see where the test failed.



At this point, you have created a test and a stub that you will modify so that the test passes.

**After every change, make all the tests pass**

1. In **MyMath\Rooter.cs**, improve the code of SquareRoot: C#

public double SquareRoot(double input)

{

return input / 2;

}

1. In Test Explorer, choose **Run All**. The code builds and the test runs. The test passes.



**Add tests to extend the range of inputs**

****

1. To improve your confidence that your code works in all cases, add tests that try a broader range of input values.
2. In your test class, add the following test, which tries a range of input values:
3. C#

[TestMethod]

public void RooterValueRange()

{

* Create an instance to test: Rooter rooter = new Rooter();
* Try a range of values:

for (double expectedResult = 1e-8;

expectedResult < 1e+8;

expectedResult = expectedResult \* 3.2)

{

RooterOneValue(rooter, expectedResult);

}

}

private void RooterOneValue(Rooter rooter, double expectedResult)

{

double input = expectedResult \* expectedResult; double actualResult = rooter.SquareRoot(input); Assert.AreEqual(expectedResult, actualResult,

delta: expectedResult / 1000);

}

1. In Test Explorer, choose **Run All**.

The new test fails, although the first test still passes.

To find the point of failure, select the failing test and then in the lower part of Test Explorer, select the top item of the**Stack Trace**.

1. Inspect the method under test to see what might be wrong. In the **MyMath.Rooter** class, rewrite

the code:

public double SquareRoot(double input)

{

double result = input;

double previousResult = -input;

while (Math.Abs(previousResult - result) > result / 1000)

{

previousResult = result;

result = result - (result \* result - input) / (2 \* result);

}

return result;

}

1. In Test Explorer, choose **Run All**. Both tests now pass.

**Add tests for exceptional cases**

****

1. Add a test for negative inputs: C#

[TestMethod]

public void RooterTestNegativeInputx()

{

Rooter rooter = new Rooter();

try

{

rooter.SquareRoot(-10);

}

catch (ArgumentOutOfRangeException e)

{

return;

}

Assert.Fail();

}

1. In Test Explorer, choose **Run All**.

The method under test loops, and must be canceled manually.

1. Choose **Cancel**.

The test stops after 10 seconds.

1. Fix the method code: C#

public double SquareRoot(double input)

{

if (input <= 0.0)

{

throw new ArgumentOutOfRangeException();

}

...

1. In Test Explorer, choose **Run All**. All the tests pass.