Test Driven Development

Bhanu Korremula
Sr. Programmer Analyst,
Business Systems Division,
Richland County

The only thing constant is change – Heraclitus

Requirements are constantly changing

Software is constantly evolving

To handle constant change go Agile with Test Driven Development

Topics that Will be Covered

- Test Driven Development
- Unit Testing Frameworks
- Testing Practices
- Visual Studio 2013 Unit Testing Framework
- Creating Unit Tests from scratch
- Test Explorer
- Data Driven Test
- Code Coverage
- Visual Studio 2013 Fakes Testing Framework

What is Test Driven Development?

- Wikipedia defines Test Driven Development as:

"*Test-driven development (TDD) is a software development process that relies on the repetition of a very short development cycle: first the developer writes an (initially failing) automated test case that defines a desired improvement or new function, then produces the minimum amount of code to pass that test, and finally refactors the new code to acceptable standards.*"

- Kent Beck is credited with having developed the technique

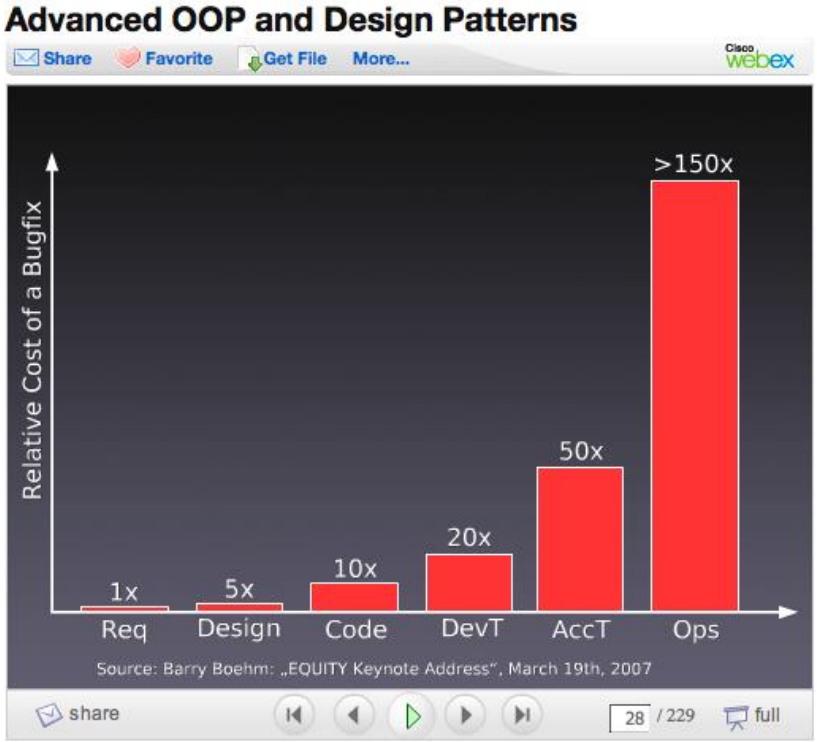
http://en.wikipedia.org/wiki/Test-driven_development

Common Reasons to Ignore Developer Testing

- Developers may think "I am the one who creates software why should I be spending my time on testing"
- Managers may think "I don't want to pay the developers to test its a tester's job"

Costs of Inadequate Testing

- Production bugs are expensive to fix
- The sooner a bug is found and fixed the cost of making and maintaining the software will go down



http://superwebdeveloper.com/2009/11/25/the-incredible-rate-of-diminishing-returns-of-fixing-software-bugs/

- People are creatures of habit some people resist change
- Don't think of Test Driven Development as whole new way of developing software
- Think of it as a way to capture various documented and undocumented tests which developers create during software development

Test Driven Development Developer Lingo

- Code that validates other code
- Small chunks of “is it working”
- By small chunks it means write unit tests
- “Never write a single line of code until you have written a failing automated test” – Kent Beck

Unit Testing Frameworks

Visual Studio Testing Framework
MS Test

NUnit

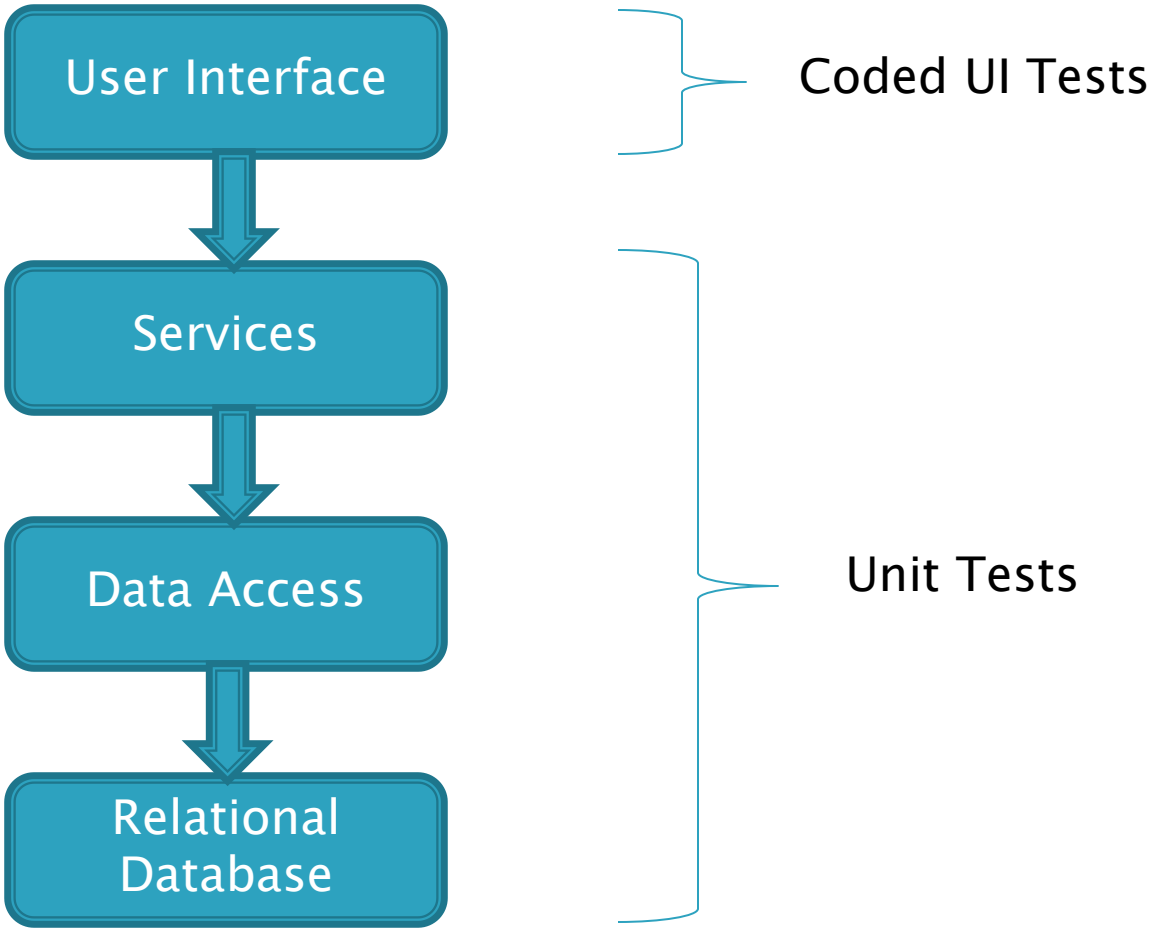
XUnit

Visual Studio Testing Framework

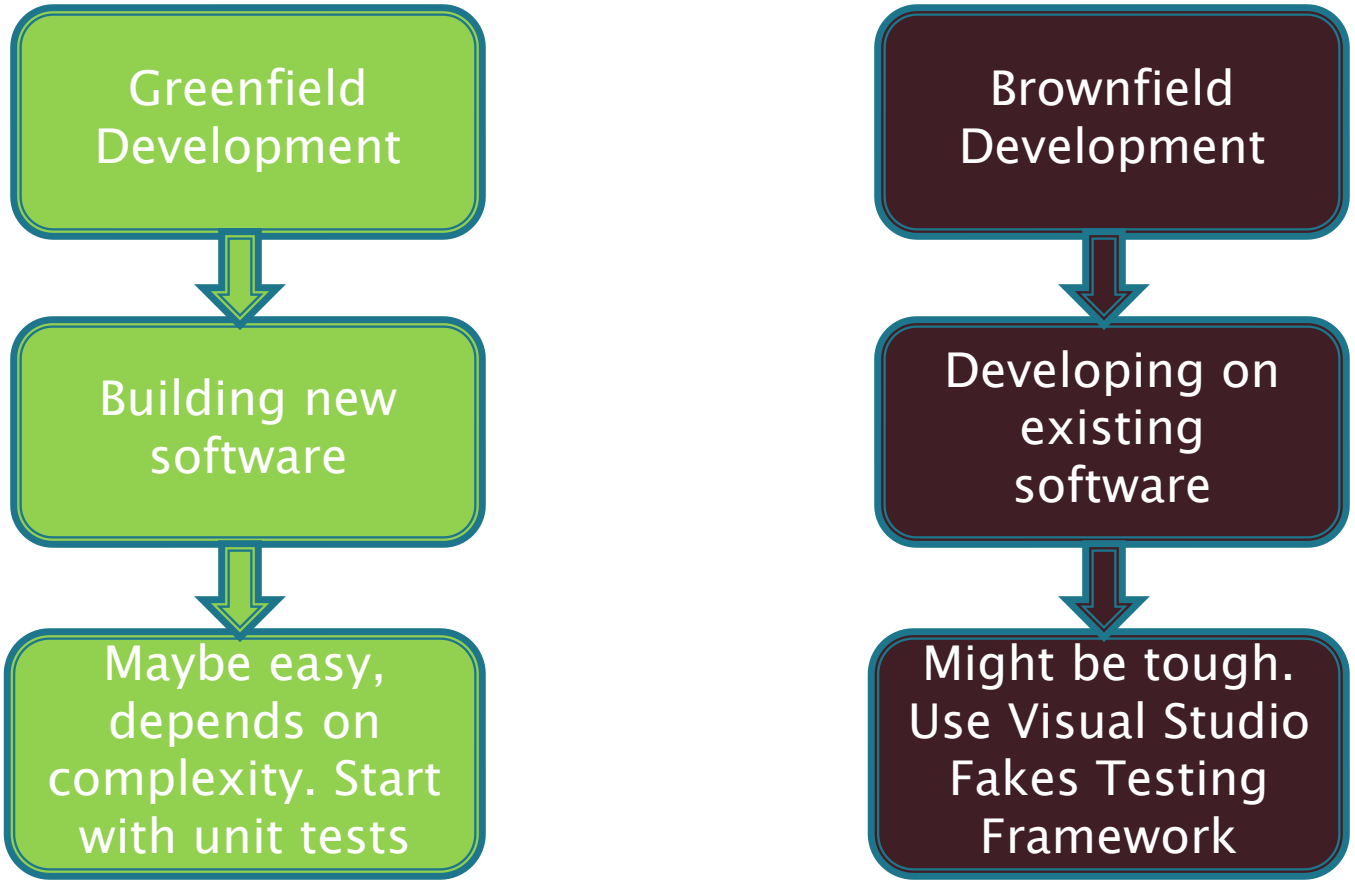
- The Visual Studio Unit Testing Framework describes Microsoft's suite of unit testing tools as integrated into some versions of Visual Studio 2005 and later.
- The Unit Testing Framework is defined in Microsoft.VisualStudio.TestTools.UnitTesting in Visual Studio 2013.
- Unit tests created with the Unit Testing Framework can be executed in Visual Studio or, using MSTest.exe, from a command line

http://en.wikipedia.org/wiki/Visual_Studio_Unit_Testing_Framework

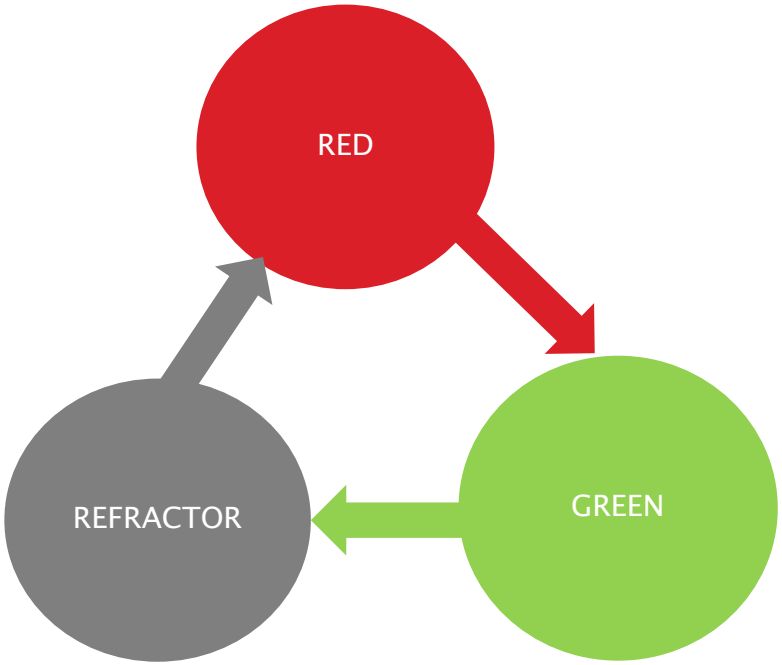
N Tier App and Testing



Testing Types for different software models



Testing Practices



- Red green refractor cycle
- Developer should create a failing test and make the bug pass
- Developers have to test classes, methods or functions at the API level
- Developers should strive to test code as it is being built

Main Elements of Visual Studio Testing Framework

Test Class	Test classes are declared as such by decorating a class with the TestClass attribute. The attribute is used to identify classes that contain test methods.
Test Method	Test methods are declared as such by decorating a unit test method with the TestMethod attribute. The attribute is used to identify methods that contain unit test code.
Assertions	An assertion is a piece of code that is run to test a condition or behavior against an expected result.
Initialization and Cleanup methods	Initialization and cleanup methods are used to prepare unit tests before running and cleaning up after unit tests have been executed.

http://en.wikipedia.org/wiki/Visual_Studio_Unit_Testing_Framework

Asserts

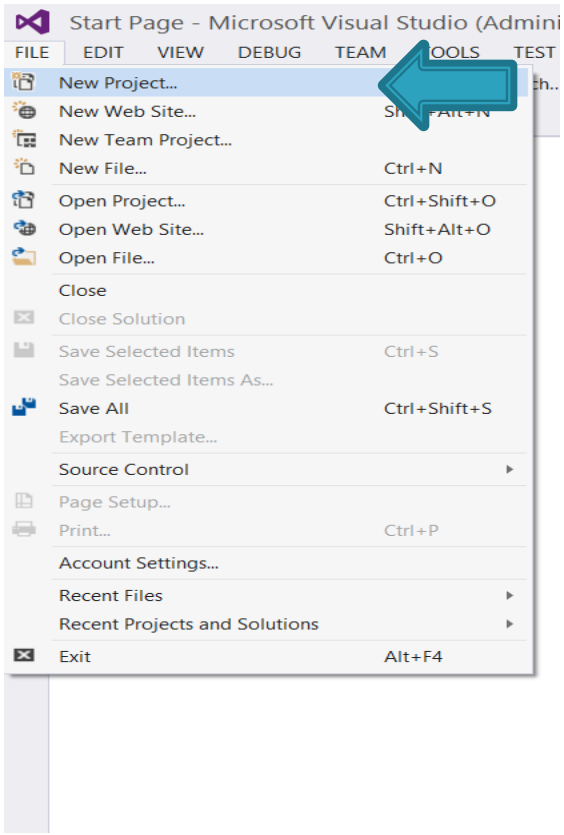
- Verifies conditions in unit tests using true/false propositions
- Normally used to compare expected and actual value

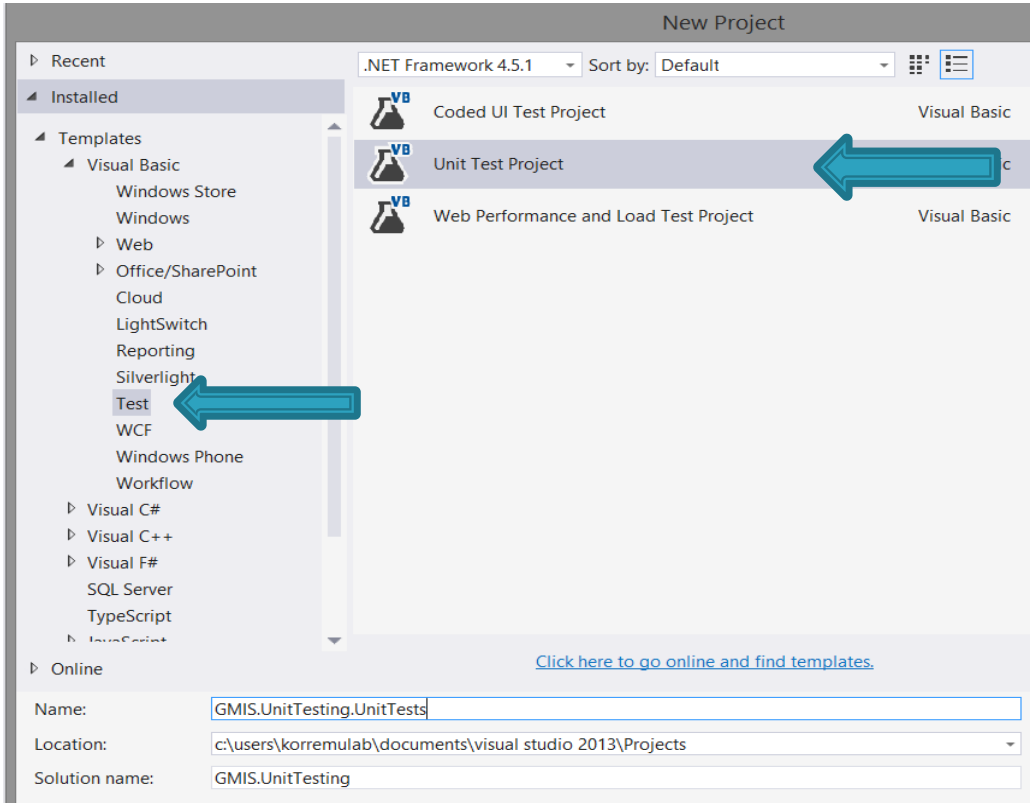
▲ Methods

	Name	Description
	AreEqual(Object, Object)	Verifies that two specified objects are equal. The assertion fails if the objects are not equal.
	AreEqual(Double, Double, Double)	Verifies that two specified doubles are equal, or within the specified accuracy of each other. The assertion fails if they are not within the specified accuracy of each other.
	AreEqual(Object, Object, String)	Verifies that two specified objects are equal. The assertion fails if the objects are not equal. Displays a message if the assertion fails.
	AreEqual(Single, Single, Single)	Verifies that two specified singles are equal, or within the specified accuracy of each other. The assertion fails if they are not within the specified accuracy of each other.
	AreEqual(String, String, Boolean)	Verifies that two specified strings are equal, ignoring case or not as specified. The assertion fails if they are not equal.
	AreEqual(Double, Double, Double, String)	Verifies that two specified doubles are equal, or within the specified accuracy of each other. The assertion fails if they are not within the specified accuracy of each other. Displays a message if the assertion fails.
	AreEqual(Object, Object, String, Object())	Verifies that two specified objects are equal. The assertion fails if the objects are not equal. Displays a message if the assertion fails, and applies the specified formatting to it.
	AreEqual(Single, Single, Single, String)	Verifies that two specified singles are equal, or within the specified accuracy of each other. The assertion fails if they are not within the specified accuracy of each other. Displays a message if the assertion fails.
	AreEqual(String, String, Boolean, CultureInfo)	Verifies that two specified strings are equal, ignoring case or not as specified, and using the culture info specified. The assertion fails if they are not equal.
	AreEqual(String, String, Boolean, String)	Verifies that two specified strings are equal, ignoring case or not as specified. The assertion fails if they are not equal. Displays a message if the assertion fails.
	AreEqual(Double, Double, Double, String, Object())	Verifies that two specified doubles are equal, or within the specified accuracy of each other. The assertion fails if they are not within the specified accuracy of each other. Displays a message if the assertion fails, and applies the specified formatting to it.
	AreEqual(Single, Single, Single, String, Object())	Verifies that two specified singles are equal, or within the specified accuracy of each other. The assertion fails if they are not within the specified accuracy of each other. Displays a message if the assertion fails, and applies the specified formatting to it.
	AreEqual(String, String, Boolean, CultureInfo, String)	Verifies that two specified strings are equal, ignoring case or not as specified, and using the culture info specified. The assertion fails if they are not equal. Displays a message if the assertion fails.
	AreEqual(String, String, Boolean, String, Object())	Verifies that two specified strings are equal, ignoring case or not as specified. The assertion fails if they are not equal. Displays a message if the assertion fails, and applies the specified formatting to it.

http://msdn.microsoft.com/en-us/library/microsoft.visualstudio.testtools.unittesting.assert.aspx

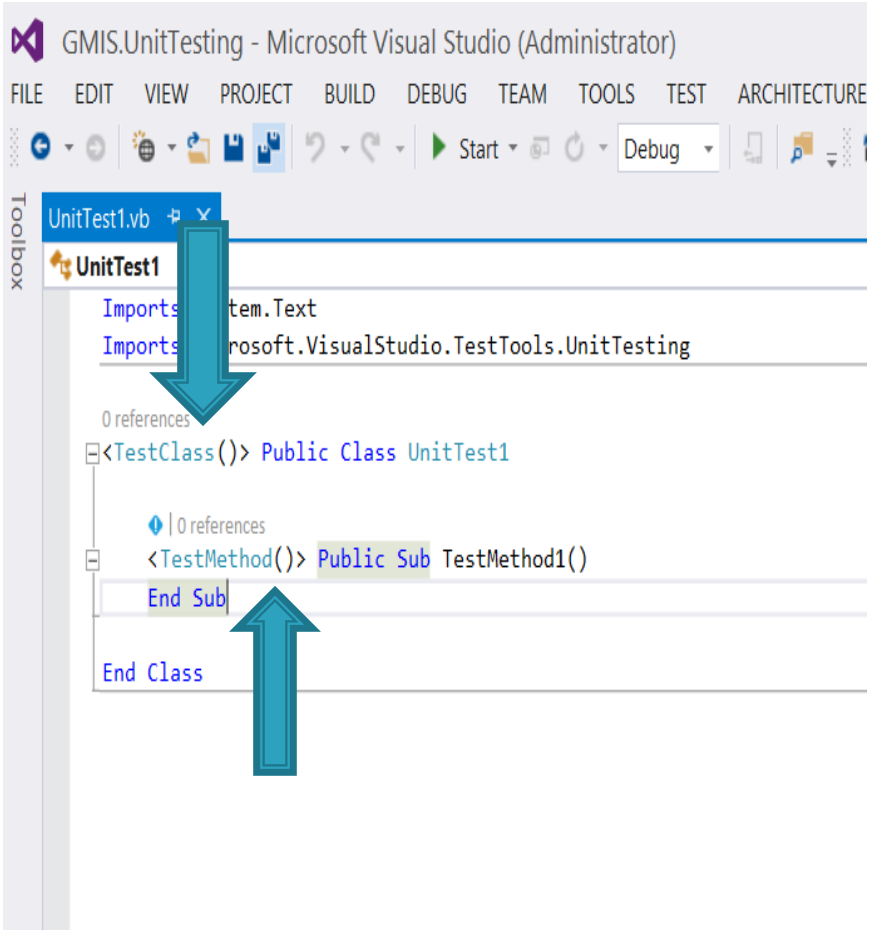
Create a Unit Test Project





Go to New Project → Visual Basic → Test → Unit Test Project to create one

What Does Default Unit Test Template Contain?



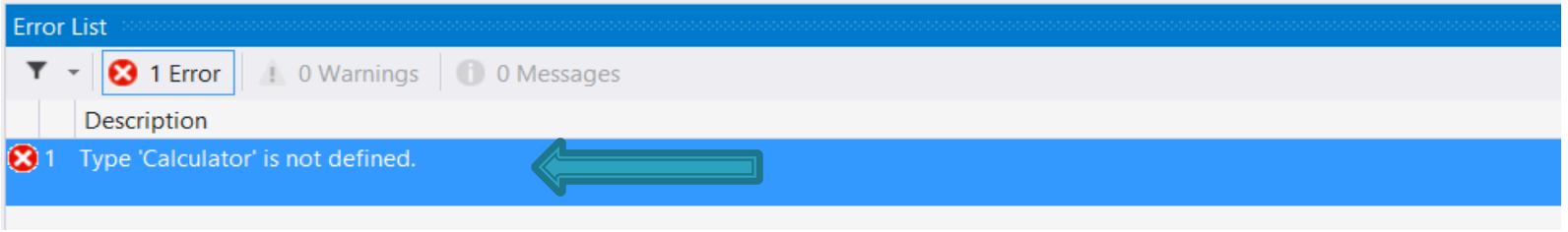
- The unit test project by default comes with test class and test method/function
- The project has two main attributes first one is <TestClass()> and second is <TestMethod()>
- <TestClass()> informs the .NET that class will have test functions
- <TestMethod()> informs .NET to execute the unit test function

Code The Test First

```
Imports System.Text
Imports Microsoft.VisualStudio.TestTools.UnitTesting

0 references
<TestClass()> Public Class CalculatorTestFixture

    0 references
    <TestMethod()> Public Sub AddTest()
        Dim calc As New Calculator
        'Dim calc As New Calculator
        Dim expected As Integer = 5
        Dim actual As Integer = 0
        actual = calc.Add(2, 3)
        Assert.AreEqual(expected, actual, "Did not get the right value from the add method")
    End Sub

End Class
```

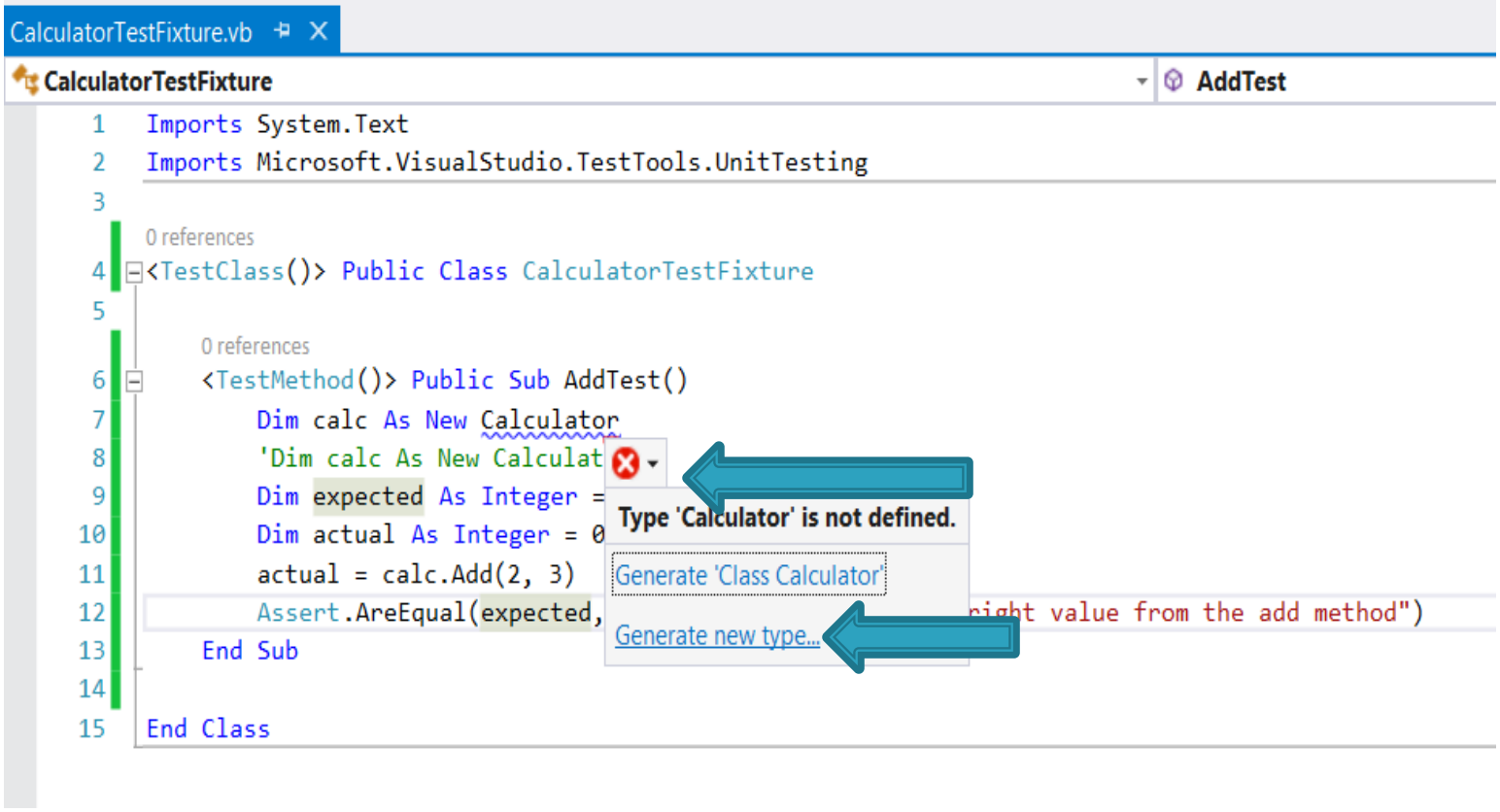
Build the Test Project

```
1   Imports System.Text
2   Imports Microsoft.VisualStudio.TestTools.UnitTesting
3
    0 references
4   <TestClass()> Public Class CalculatorTestFixture
5
        0 references
6       <TestMethod()> Public Sub AddTest()
7           Dim calc As New Calculator
8           'Dim calc As New Calculator
9           Dim expected As Integer = 5
10          Dim actual As Integer = 0
11          actual = calc.Add(2, 3)
12          Assert.AreEqual(expected, actual, "Did not get the right value from the add method")
13      End Sub
14
15  End Class
```



After the build we will get a compilation of the errors

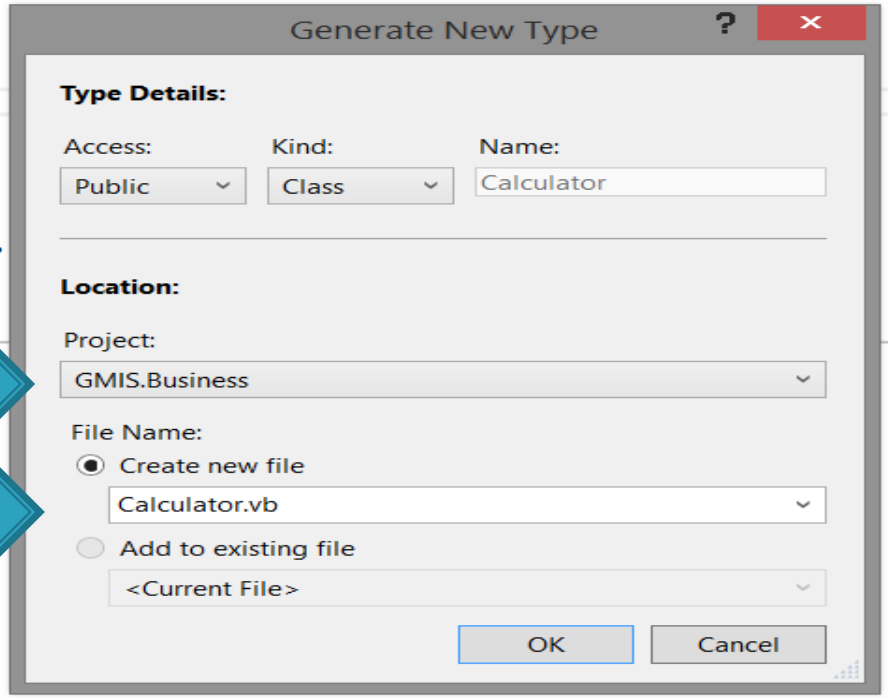
Generate The Class From Unit Test



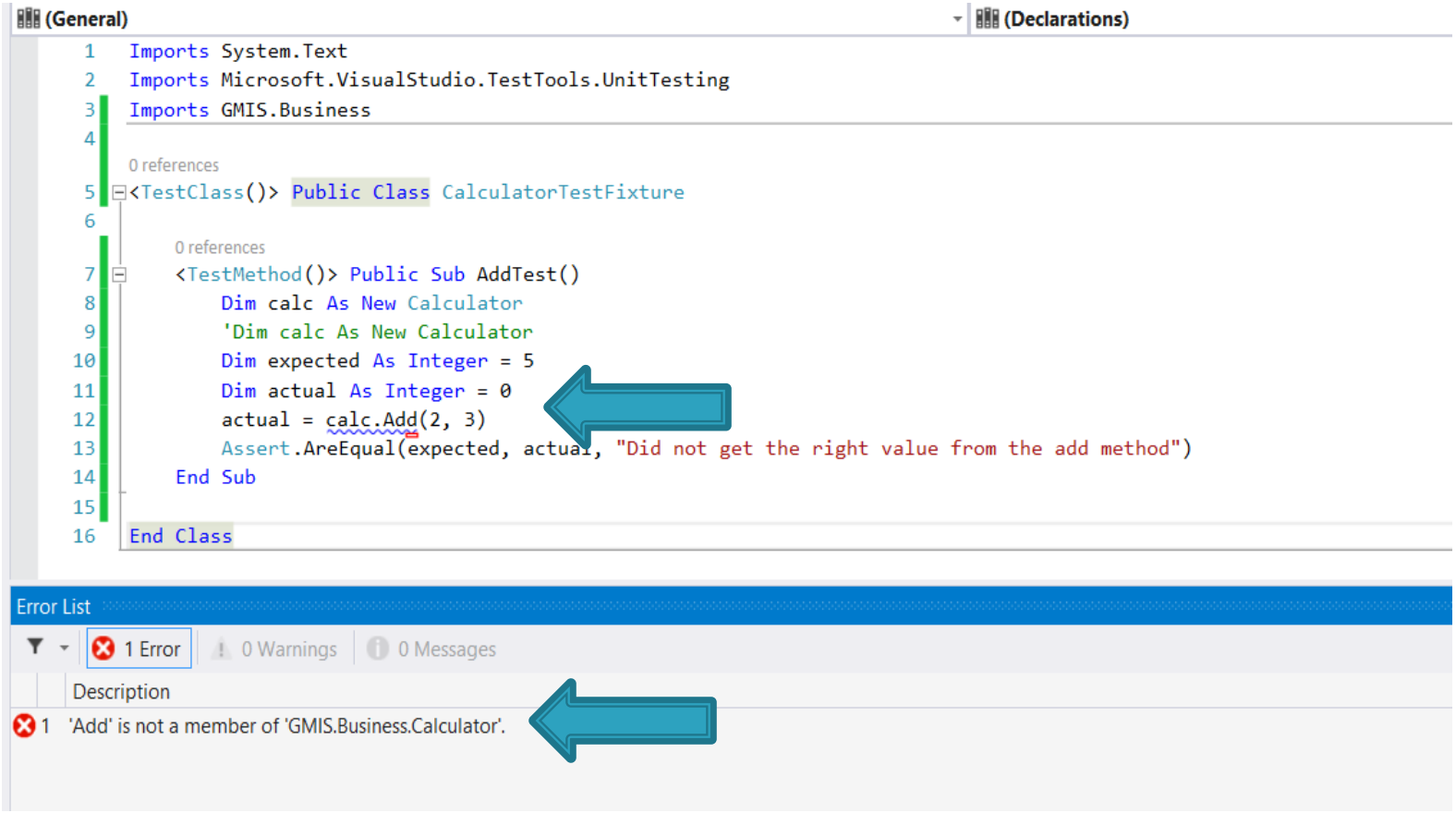
Click on generate new type

Select Appropriate Project for Class Generation

```
1   Imports System.Text
2   Imports Microsoft.VisualStudio.TestTools.UnitTesting
3
        0 references
4   <TestClass()> Public Class CalculatorTestFixture
5
            0 references
6       <TestMethod()> Public Sub AddTest()
7           Dim calc As New Calculator
8           'Dim calc As New Calculator
9           Dim expected As Integer = 5
10          Dim actual As Integer = 0
11          actual = calc.Add(2, 3)
12          Assert.AreEqual(expected, actual,
13      End Sub
14
15  End Class
```

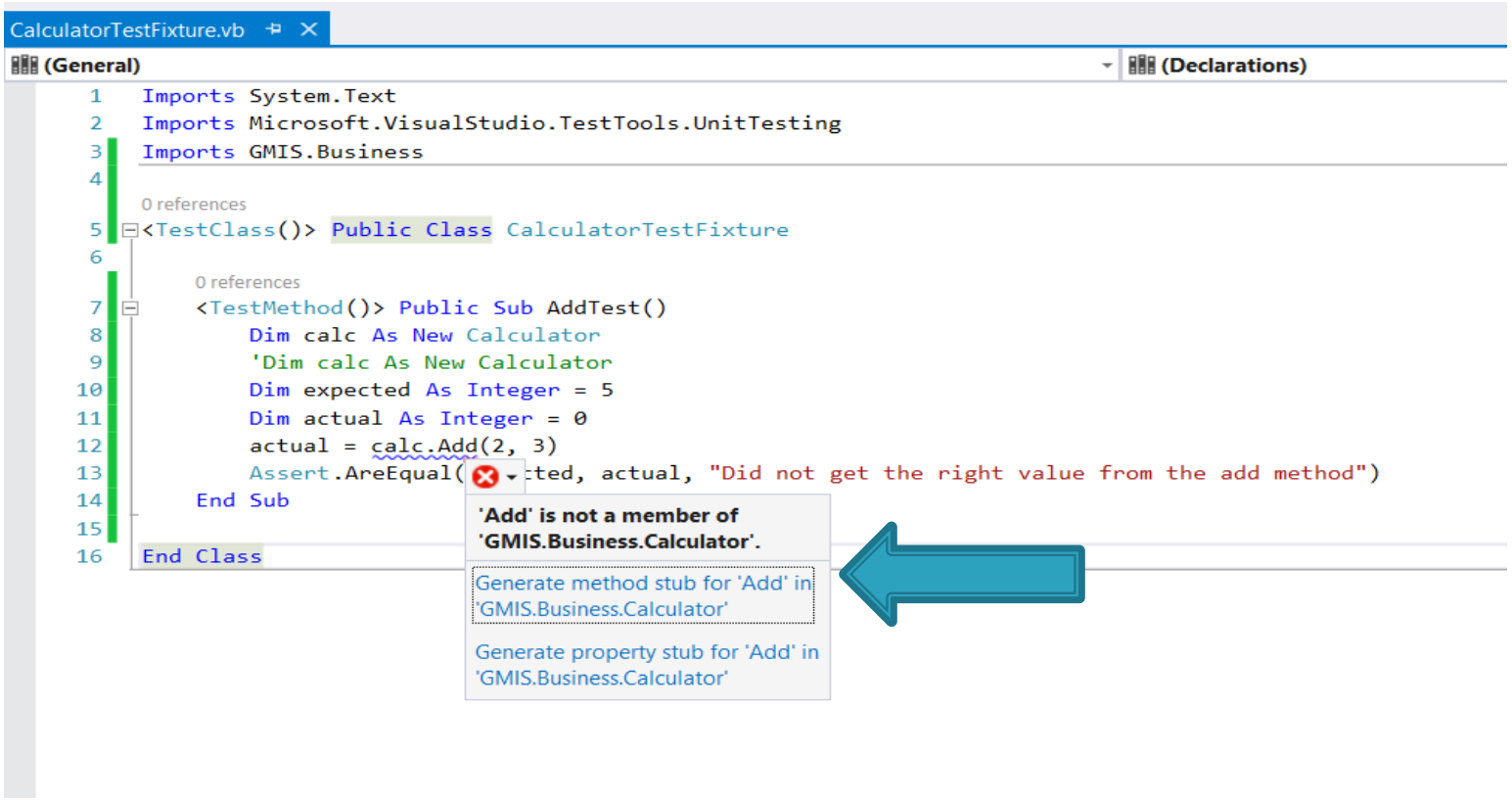


Build Again



Build breaks at the function which has not implemented yet

Generate The Function Code From Unit Test



Generate function code and build

Rebuild the Solution

Calculator.vb | CalculatorTestFixture.vb

CalculatorTestFixture | AddTest

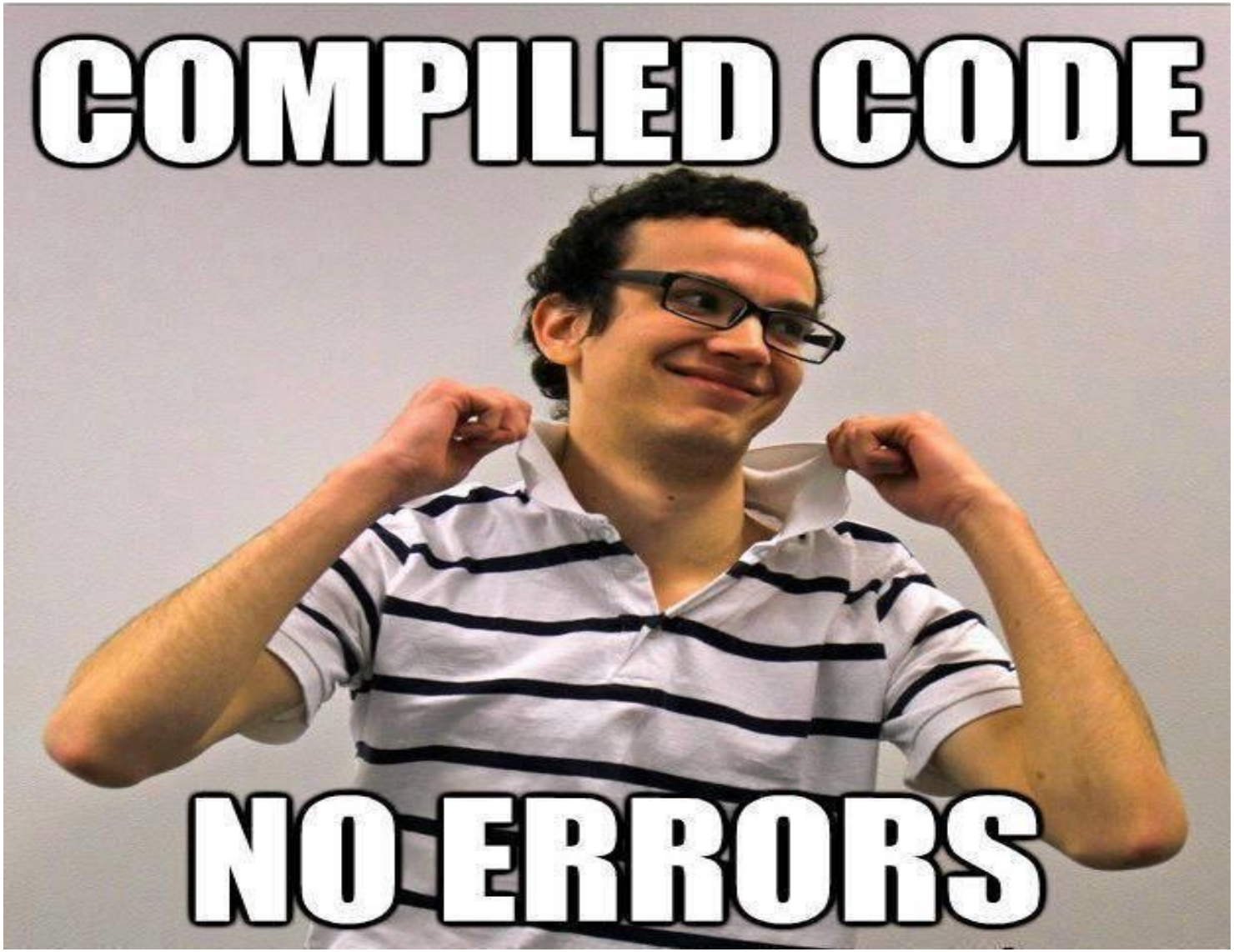
```
1   Imports System.Text
2   Imports Microsoft.VisualStudio.TestTools.UnitTesting
3   Imports GMIS.Business
4
    0 references
5   <TestClass()> Public Class CalculatorTestFixture
6
        0 references
7       <TestMethod()> Public Sub AddTest()
8           Dim calc As New Calculator
9           'Dim calc As New Calculator
10          Dim expected As Integer = 5
11          Dim actual As Integer = 0
12          actual = calc.Add(2, 3)
13          Assert.AreEqual(expected, actual, "Did not get the right value from the add method")
14      End Sub
15
16  End Class
```

Output

Show output from: Build

```
------ Build started: Project: GMIS.Business, Configuration: Debug Any CPU ------
  GMIS.Business -> C:\Users\korremulab\documents\visual studio 2013\Projects\GMIS.UnitTesting\GMIS.Business\bin\Debug\GMIS.Business.dll
------ Build started: Project: GMIS.UnitTesting.UnitTests, Configuration: Debug Any CPU ------
  GMIS.UnitTesting.UnitTests -> C:\Users\korremulab\documents\visual studio 2013\Projects\GMIS.UnitTesting\GMIS.UnitTesting.UnitTests\bin\Debug\GMIS.UnitTesting.UnitTests.dll
========== Build: 2 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

Builds successfully


COMPILED CODE
NO ERRORS

Setup of the Test Class

```
<TestClass()>
0 references
Public Class CalculatorTestFixture

    Private testContextInstance As TestContext

    '''<summary>
    '''Gets or sets the test context which provides
    '''information about and functionality for the current test run.
    '''</summary>
    0 references
    Public Property TestContext() As TestContext
        Get
            Return testContextInstance
        End Get
        Set(ByVal value As TestContext)
            testContextInstance = value
        End Set
    End Property
```

TextContext() used to come as default in test class

```
Private m_SystemUnderTest As Calculator

'''<summary>
'''Gets or sets the test context which provides
'''information about and functionality for the current test run.
'''</summary>
4 references | 2/4 passing
Public ReadOnly Property SystemUnderTest() As Calculator
    Get
        If m_SystemUnderTest Is Nothing Then
            m_SystemUnderTest = New Calculator()
        End If
        Return m_SystemUnderTest
    End Get

End Property
```

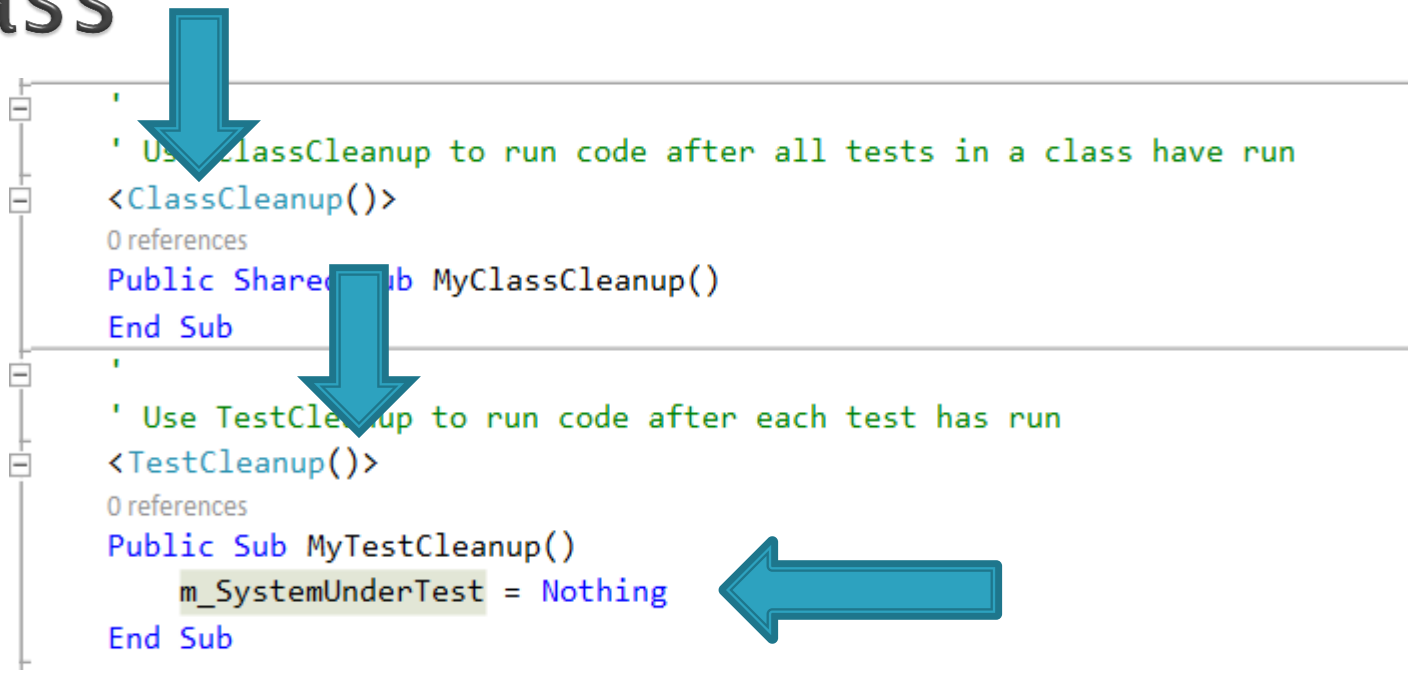
Replace TextContext() with the common class declaration to reduce code

Setup of the Test Class

```
#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()>
    0 references
    Public Shared Sub MyClassInitialize(ByVal testContext As TestContext)
    End Sub
    '
    ' Use TestInitialize to run code before running each test
    <TestInitialize()>
    0 references
    Public Sub MyTestInitialize()
    End Sub
```

- Functions marked with the <ClassInitialize()> attribute will run before the test class
- Functions marked with <TestInitialize()> attribute will run before each test

Cleanup/Teardown of the Test Class

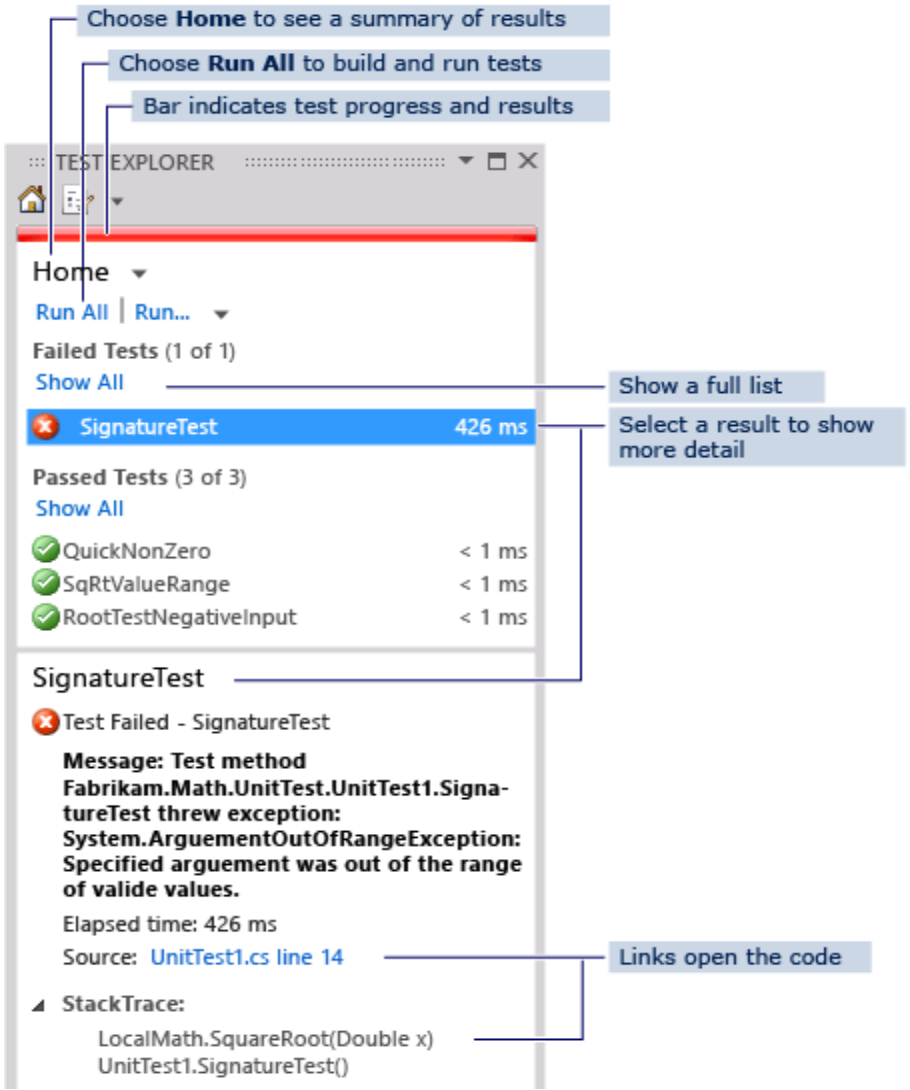


```
'
' Use ClassCleanup to run code after all tests in a class have run
<ClassCleanup()>
0 references
Public Shared Sub MyClassCleanup()
End Sub
'
' Use TestCleanup to run code after each test has run
<TestCleanup()>
0 references
Public Sub MyTestCleanup()
    m_SystemUnderTest = Nothing
End Sub
```

- Functions marked with the <ClassCleanup()> attribute will run after the test class
- Functions marked with <TestCleanup()> attribute will run after each test

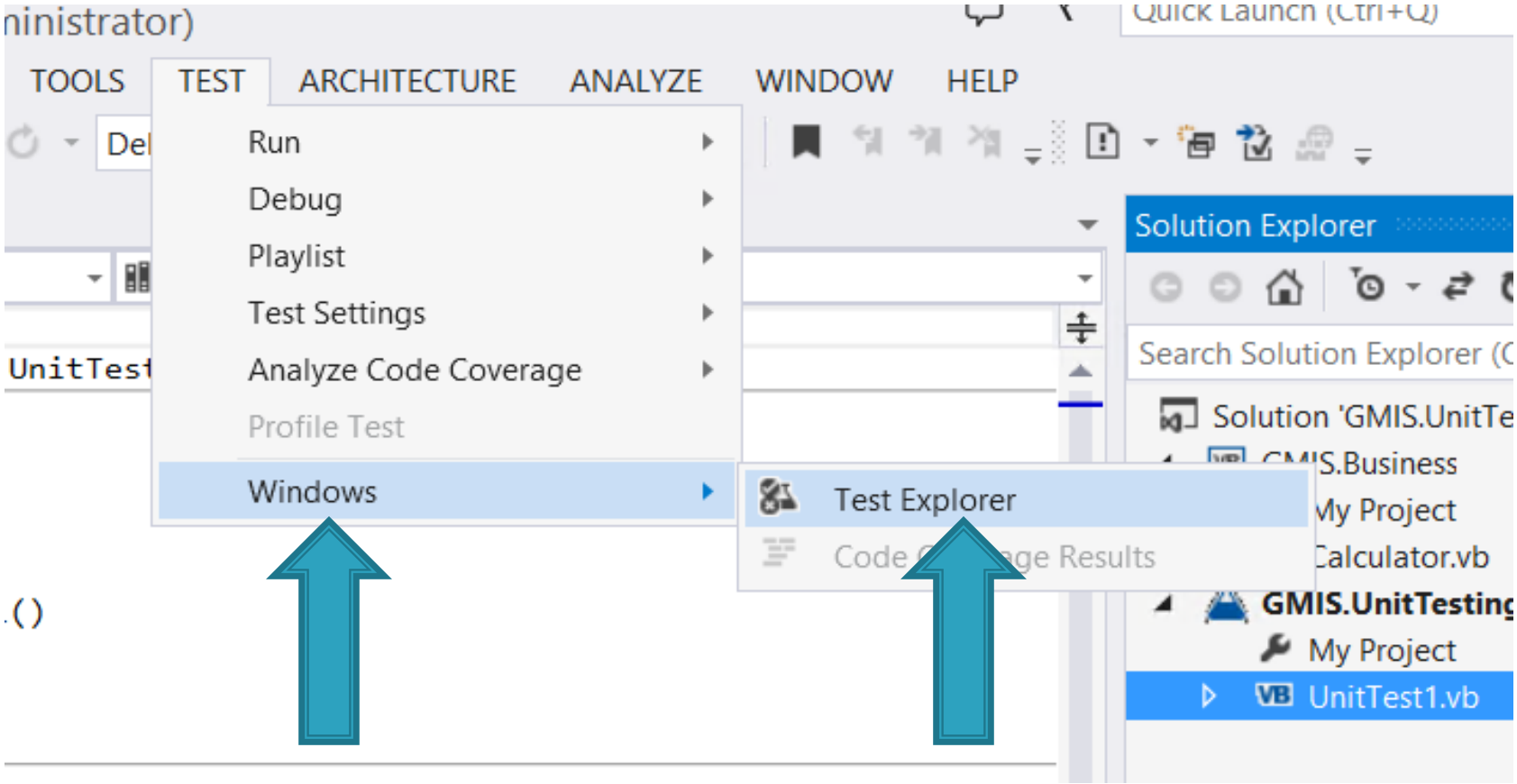
Features of Test Explorer

- Run unit tests
- Debug tests
- Identify long running tests
- Group tests into categories
- Code coverage

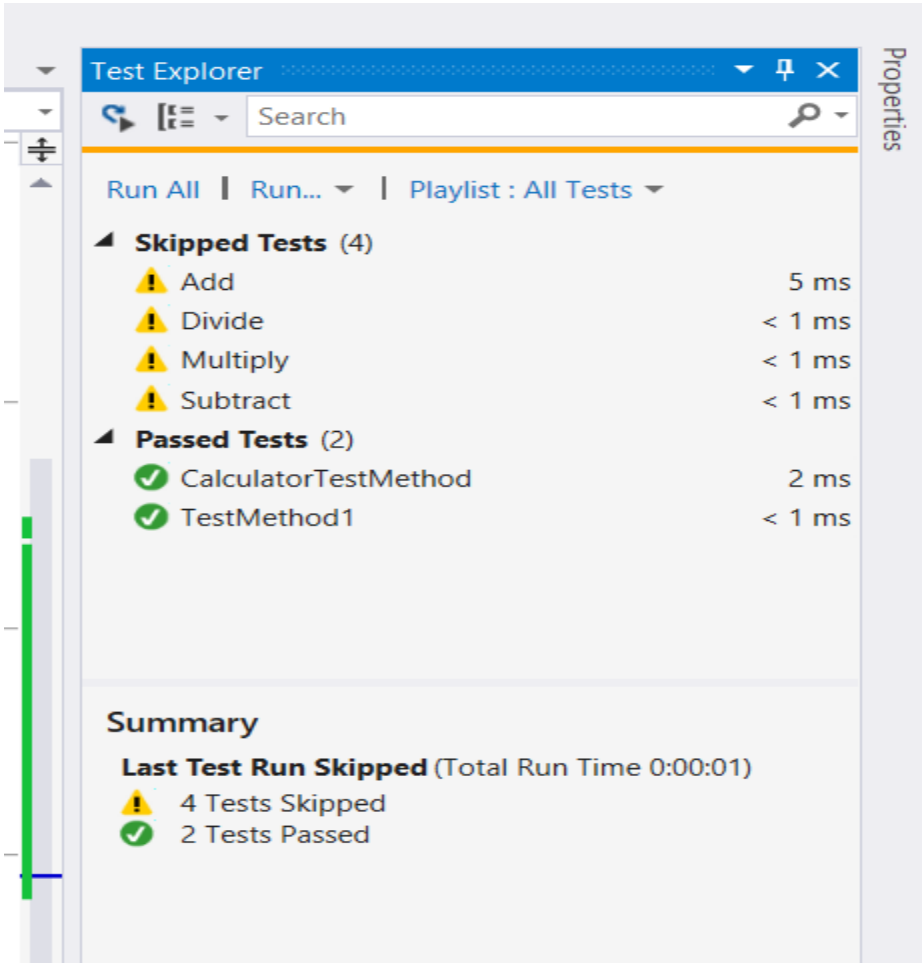


http://msdn.microsoft.com/en-us/library/hh212233.aspx

How to Get to Test Explorer?

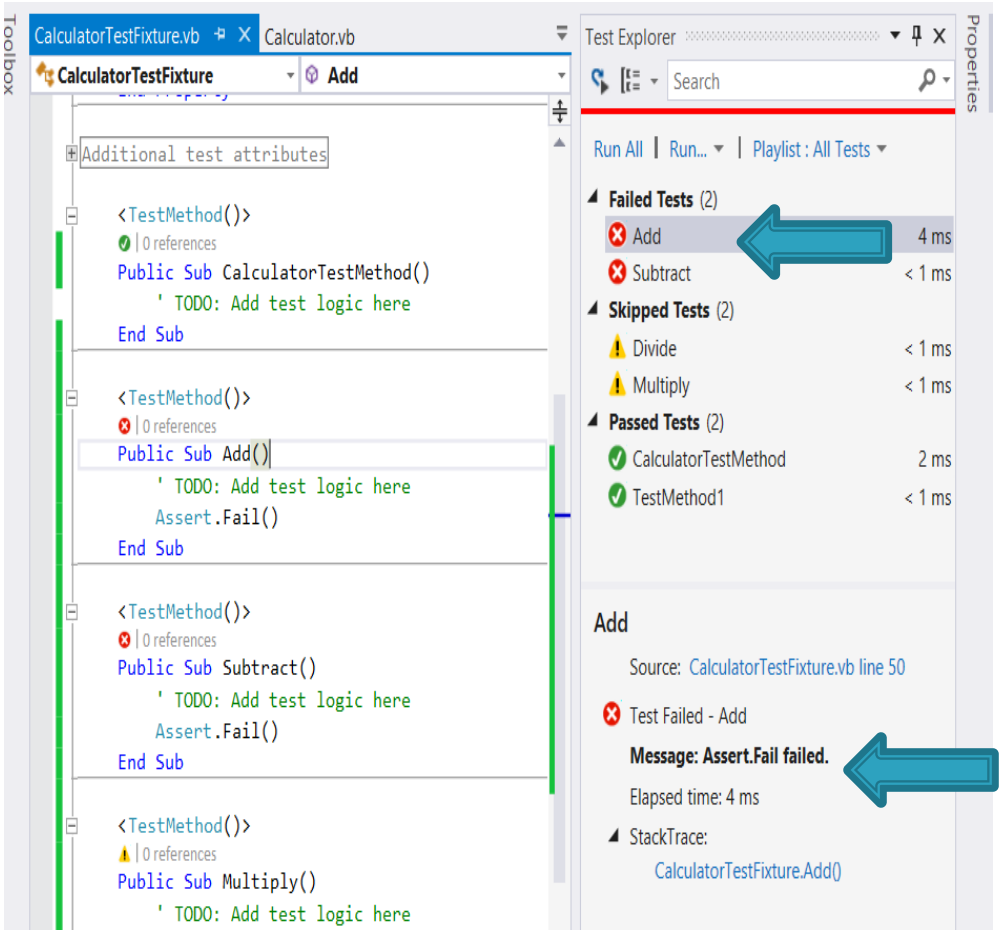


Test Explorer Features



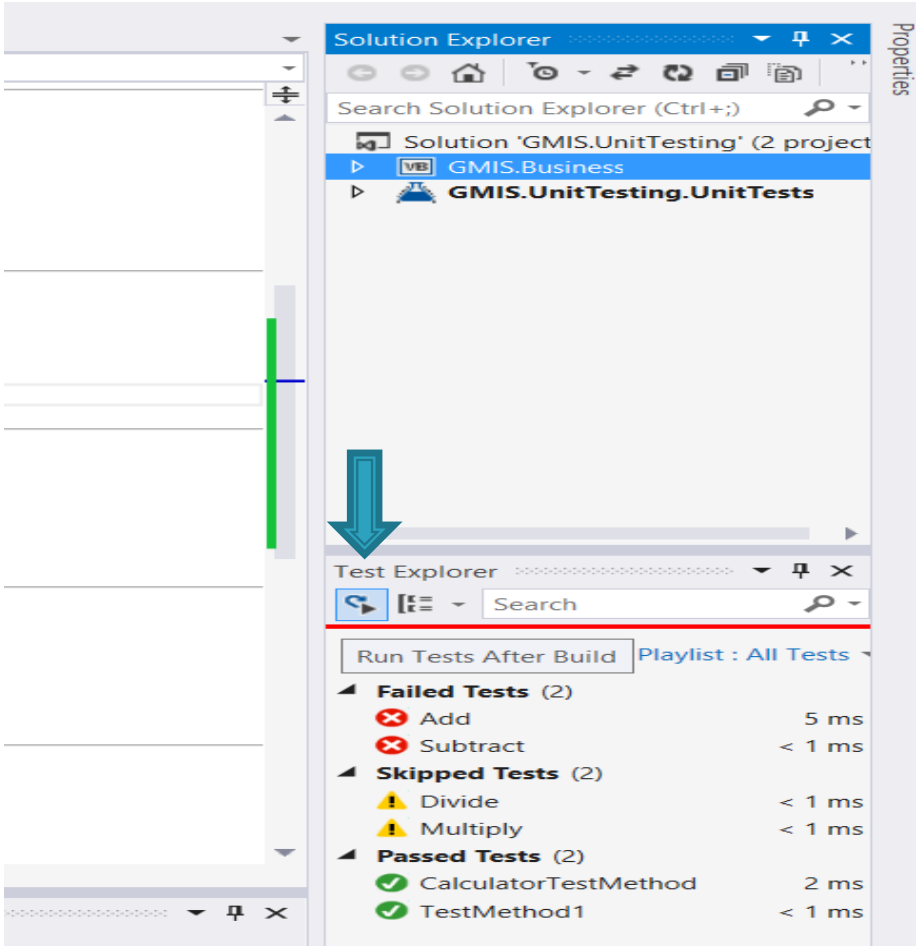
- As new tests are written they are automatically added to the test explorer
- All tests can be run with a single click

Test Explorer Identifying Failed Tests



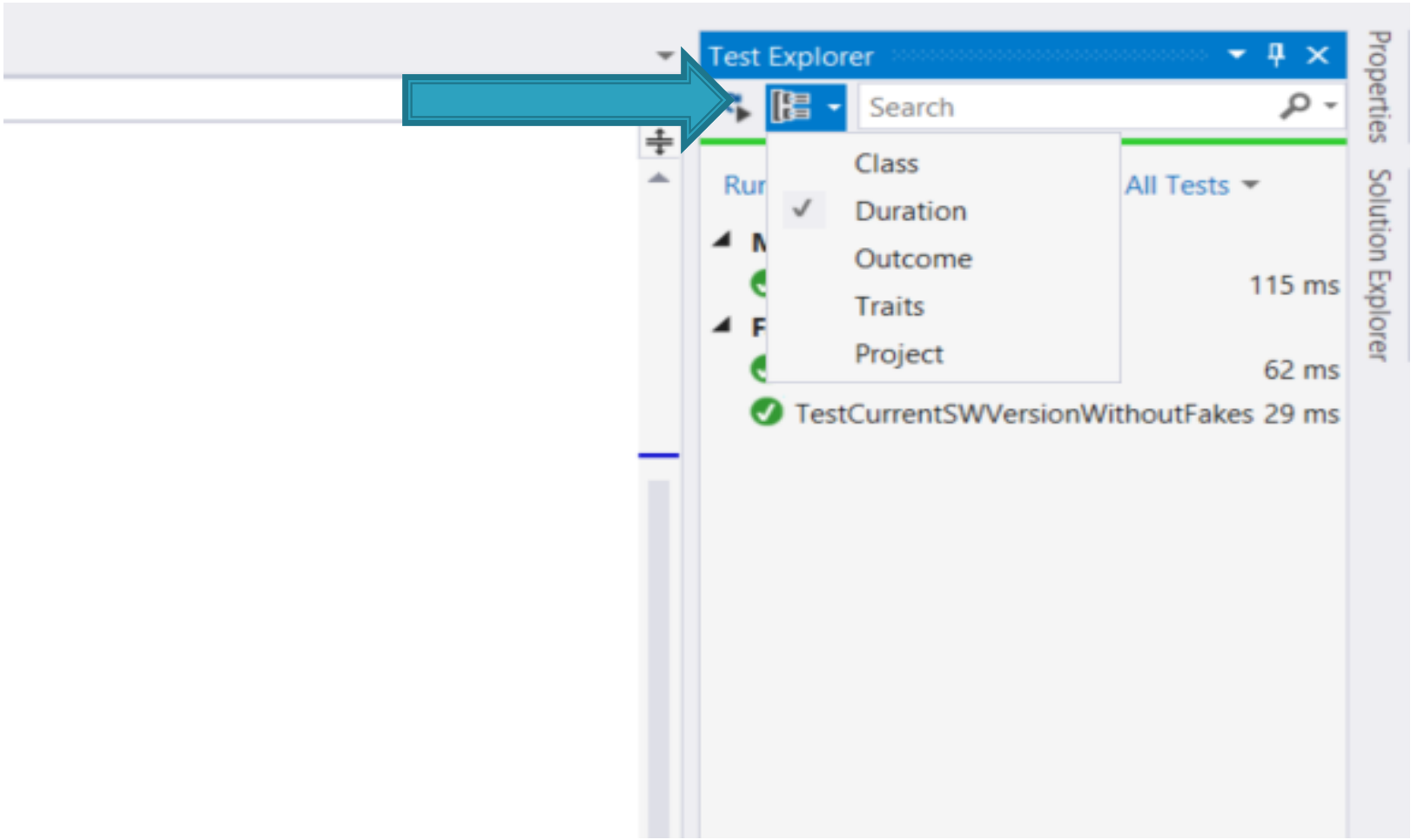
- Failing, passing and inconclusive tests can be easily identified at one glance
- With a single click one can go to the code of the failed test case and also look at the stack trace
- Developer can also run just one failed test

Running Tests After the Build

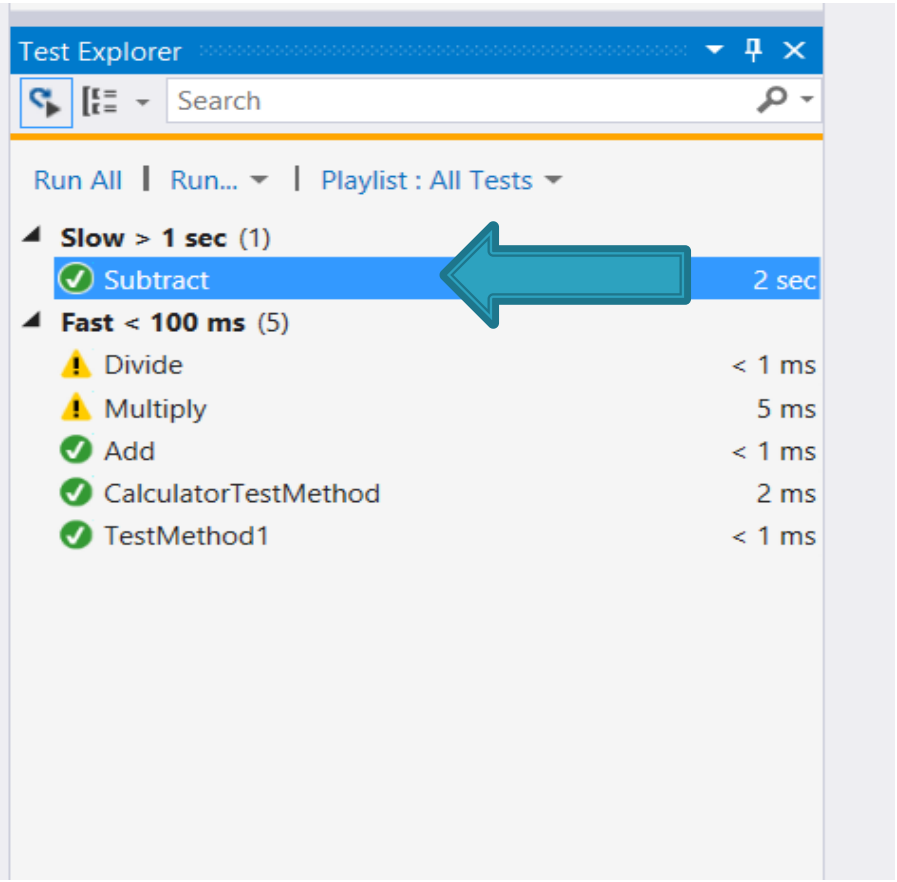


- Tests can also be set to run immediately after the build by clicking on the button shown in the picture.

Filtering By Various Test Categories

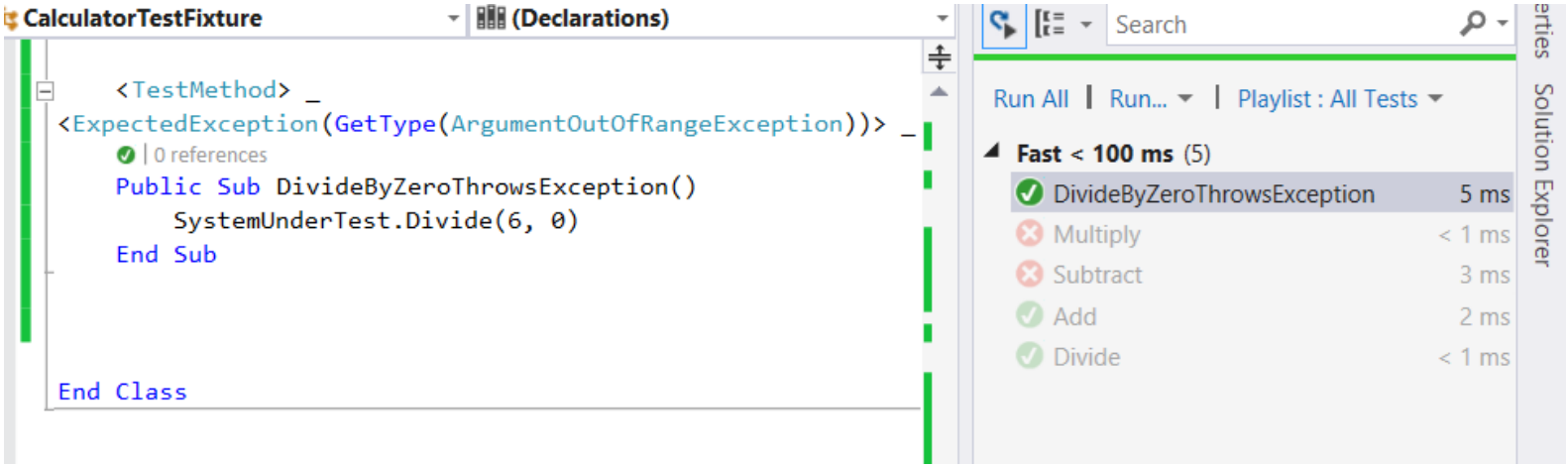


Identify Long Running Tests



- Some tests take more time
- Test explorer will help to identify those tests

Add Tests for Exceptional Cases



- Add a test which specially tests for an exception
- Added <ExpectedException(GetType(ArgumentOutofRange))> attribute to the function and test expects an exception from code

Data Driven Tests

- Unit test method can be setup to retrieve the value from the data source
- The method is run successively for each row in the data source
- This makes it easy to test a variety of input by using a single method

```
[DataSource(@"Provider=Microsoft.SqlServerCe.Client.4.0; Data Source=C:\Data\MathsData.sdf;", "Numbers")]
[TestMethod()]
public void AddIntegers_FromDataSourceTest()
{
    var target = new Maths();

    // Access the data
    int x = Convert.ToInt32(TestContext.DataRow["FirstNumber"]);
    int y = Convert.ToInt32(TestContext.DataRow["SecondNumber"]);
    int expected = Convert.ToInt32(TestContext.DataRow["Sum"]);
    int actual = target.IntegerMethod(x, y);
    Assert.AreEqual(expected, actual,
        "x:<{0}> y:<{1}>",
        new object[] {x, y});

}
```

http://msdn.microsoft.com/en-us/library/ms182527.aspx

Code Coverage

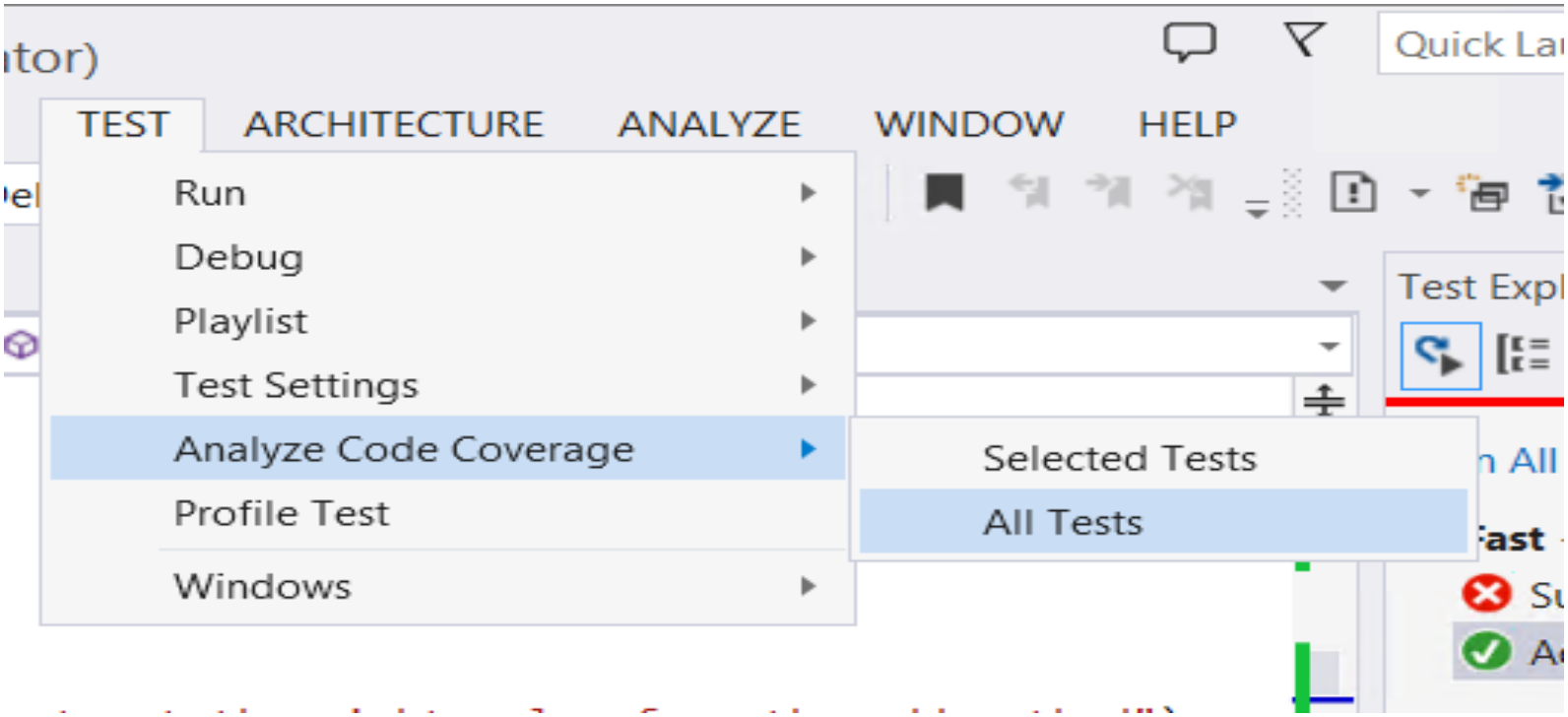
Will determine what percentage of project's code is covered by unit tests

Tests should cover a large portion of the code

Good to have more than 80 % of the code covered by the tests

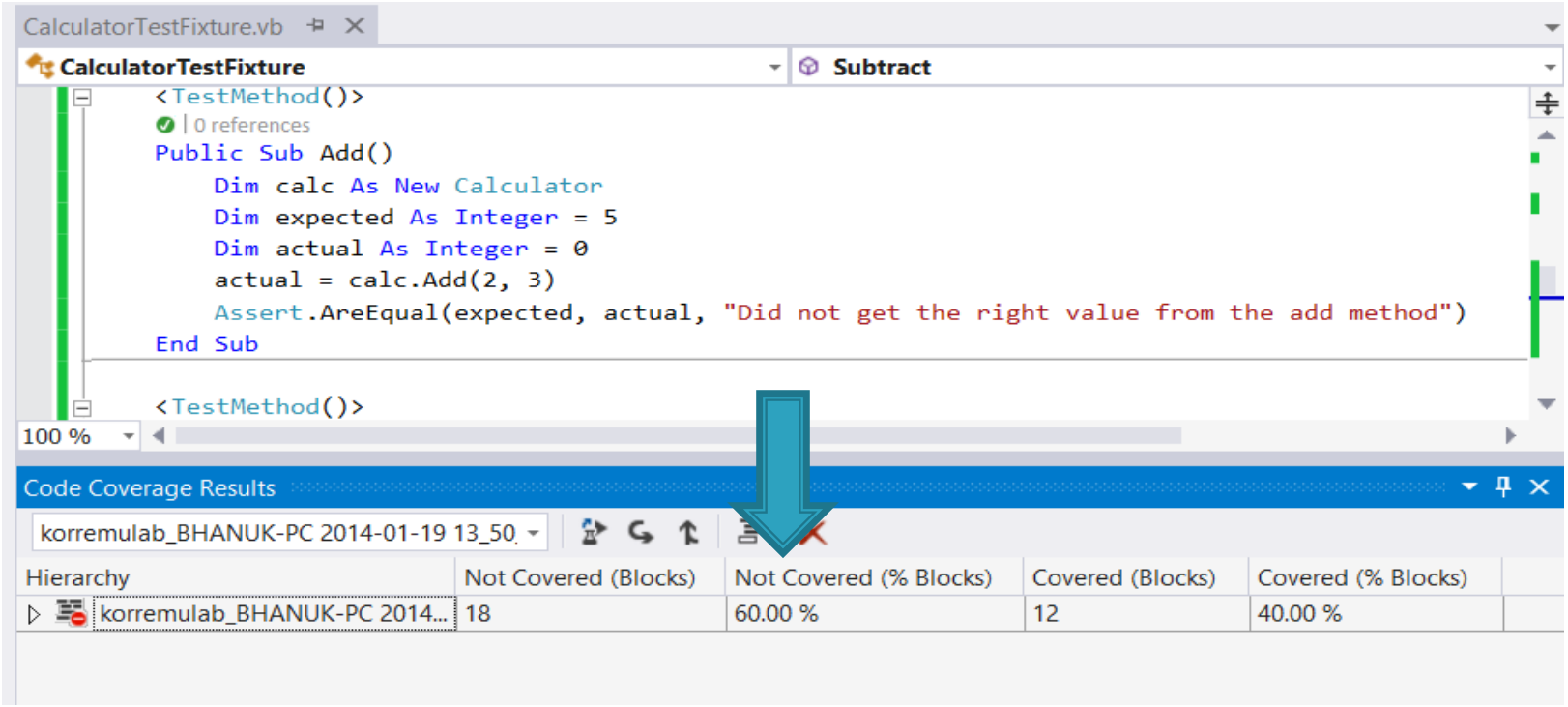
Remember just because you have 100 % code coverage you may not have working code

How to Run Code Coverage?



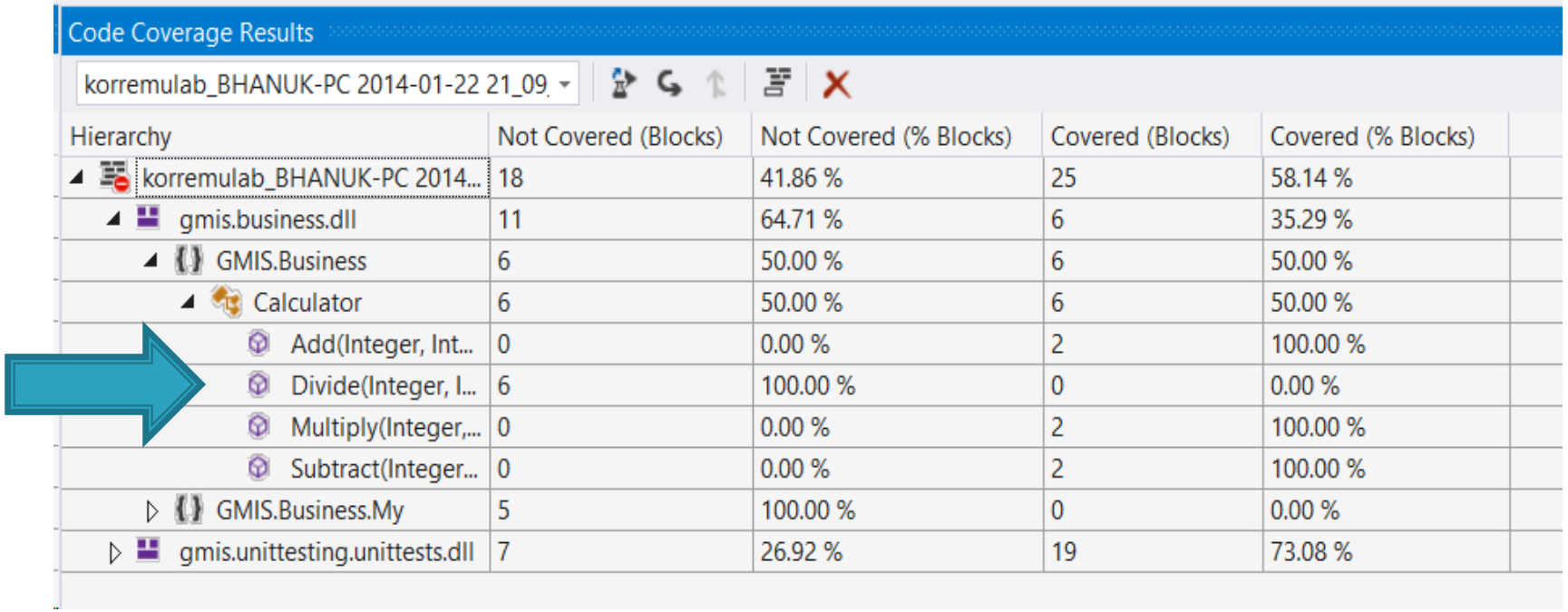
Go to Test → Analyze Code Coverage → All Tests

Code Coverage Results



Only 40 % of code is covered in the above example. The developer needs to write more tests.

Code Coverage Drilldown Results



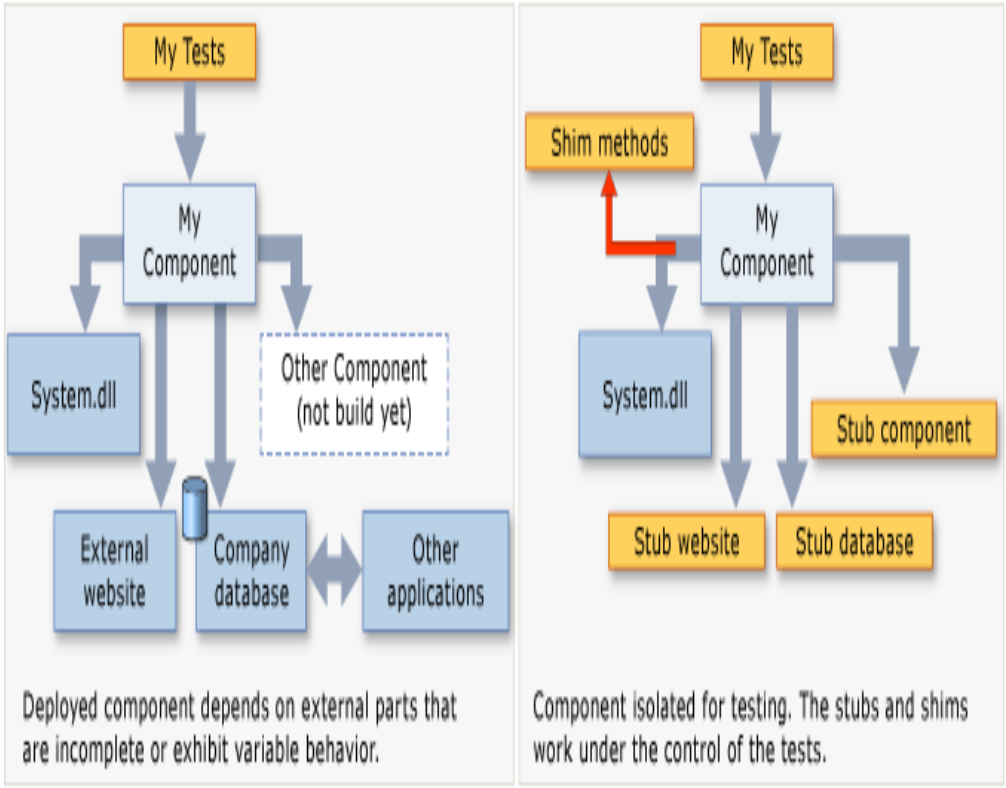
Code Coverage Results

korremulab_BHANUK-PC 2014-01-22 21_09

Hierarchy	Not Covered (Blocks)	Not Covered (% Blocks)	Covered (Blocks)	Covered (% Blocks)
korremulab_BHANUK-PC 2014...	18	41.86 %	25	58.14 %
gmis.business.dll	11	64.71 %	6	35.29 %
GMIS.Business	6	50.00 %	6	50.00 %
Calculator	6	50.00 %	6	50.00 %
Add(Integer, Int...	0	0.00 %	2	100.00 %
Divide(Integer, I...	6	100.00 %	0	0.00 %
Multiply(Integer,...	0	0.00 %	2	100.00 %
Subtract(Integer...	0	0.00 %	2	100.00 %
GMIS.Business.My	5	100.00 %	0	0.00 %
gmis.unittesting.unittests.dll	7	26.92 %	19	73.08 %

You can drill down and see which function is not covered by Unit Tests

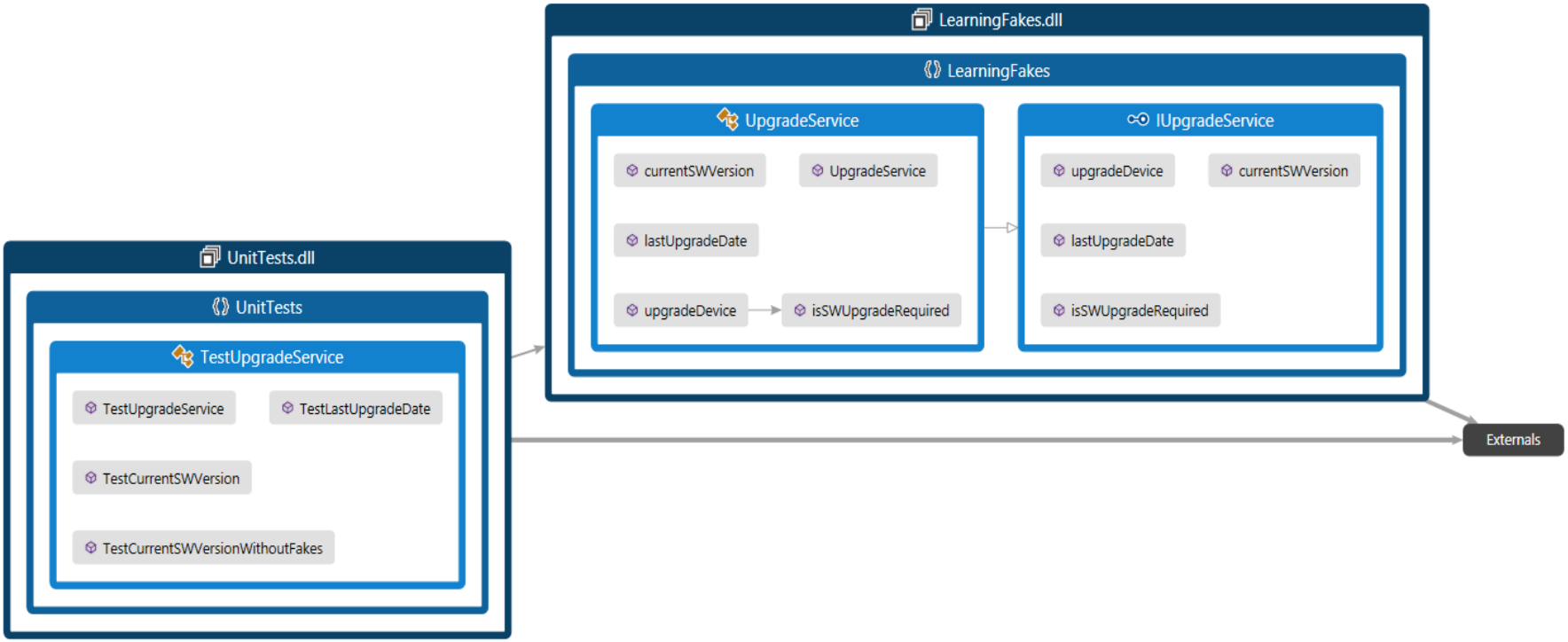
Visual Studio Fakes Testing Framework



- Fakes help you isolate the code you are testing by replacing other parts of the application with *stubs* or *shims*. These are small pieces of code that are under the control of your tests. By isolating your code for testing, you know that if the test fails, the cause is there and not somewhere else. Stubs and shims also let you test your code even if other parts of your application are not working yet.

http://msdn.microsoft.com/en-us/library/hh549175.aspx

Sample Project Dependency Diagram



http://www.codeproject.com/Articles/582812/Unit-testing-with-Fakes-with-Visual-studio-Premium

Test Class With No Fakes

```
[TestMethod]
0 references
public void TestCurrentSWVersionWithoutFakes()
{
    int expected = 5;
    UpgradeService us = new UpgradeService();
    int actual = us.currentSWVersion(1);
    Assert.AreEqual(expected, actual, "Same Versions found");

}
```

Here we are creating the instance of the class and calling its method

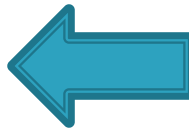
http://www.codeproject.com/Articles/582812/Unit-testing-with-Fakes-with-Visual-studio-Premium

Stubs

- A stub replaces a class with a small substitute that implements the same interface. To use stubs, you have to design your application so that each component depends only on interfaces, and not on other components.
- Creates delegate based functions to be used in the test class
- The functions have to be consumed using the anonymous and lambda functions

Test Class With Stubs

```
[TestMethod]
0 references
public void TestCurrentSWVersion()
{
    int expected = 10;
    IUpgradeService us = new LearningFakes.Fakes.StubIUpgradeService()
        {
            CurrentSWVersionInt32 = (DeviceID) => { return 10;},
            IsSWUpgradeRequiredInt32 = (DeviceID) => { return true; }
        };
    int actual = us.currentSWVersion(1);
    Assert.AreEqual(expected, actual, "Same Versions found");

}
```



Interfaces are being used
Visual Studio Fakes Testing Framework creates fake implementations of methods

http://www.codeproject.com/Articles/582812/Unit-testing-with-Fakes-with-Visual-studio-Premium

Shims

- A shim modifies the compiled code of your application at run time so that instead of making a specified method call, it runs the shim code that your test provides. Shims can be used to replace calls to assemblies that you cannot modify, such .NET assemblies.
- This is used where classes are tightly coupled and there are no interfaces

Test Class With Shim

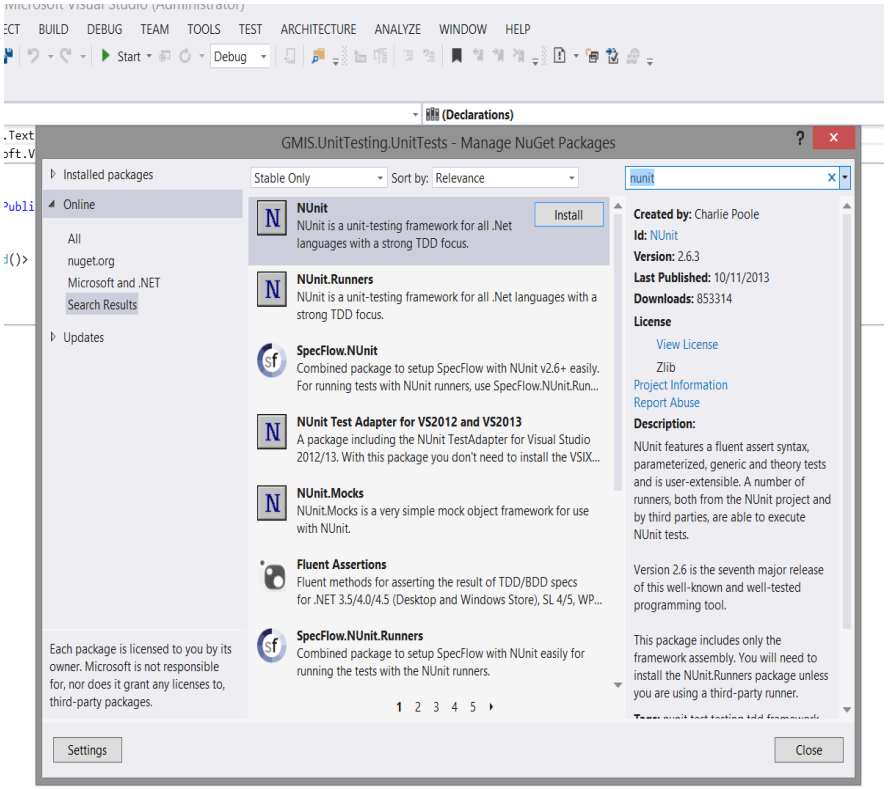
```
[TestMethod]
0 references
public void TestLastUpgradeDate()
{
    using (ShimsContext.Create())
    {
        System.Fakes.ShimDateTime.NowGet =
        () =>
        { return new DateTime(2010, 11, 5); };
        var fakeTime = DateTime.Now; // It is always DateTime(2010, 11, 5);
        Console.WriteLine(fakeTime.ToShortDateString());
    }
    var correctTime = DateTime.Now;
    Console.WriteLine(correctTime.ToShortDateString());
}
```

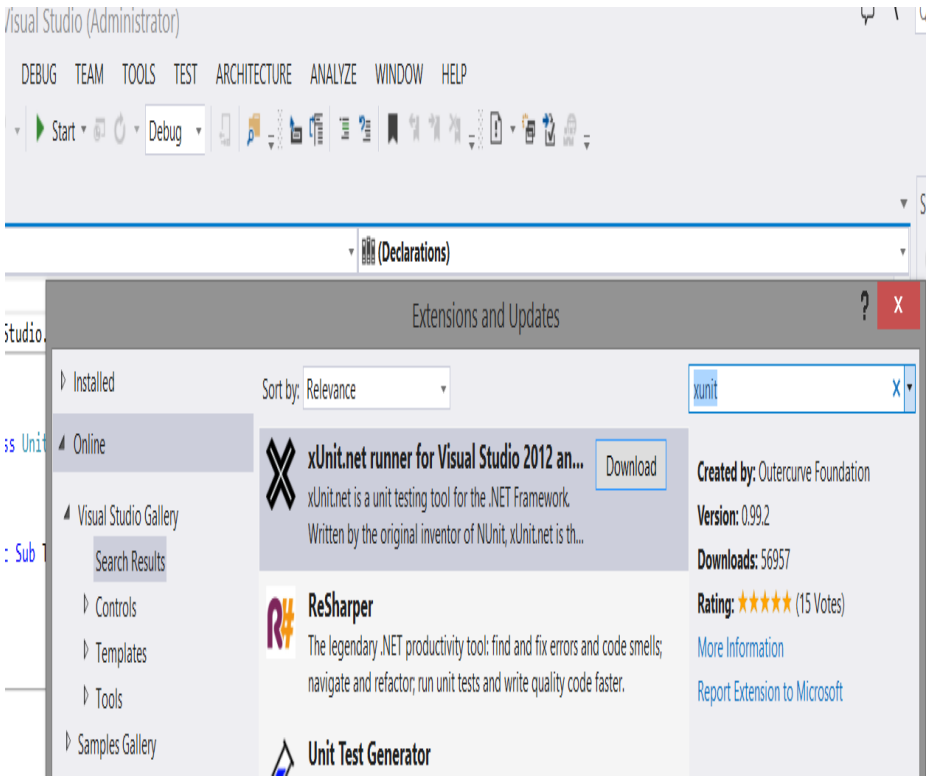
Used when code has no interfaces
Visual Studio Fakes Testing Framework creates fake implementations of .NET methods
You get the current DateTime outside the ShimsContext Scope

http://www.codeproject.com/Articles/582812/Unit-testing-with-Fakes-with-Visual-studio-Premium

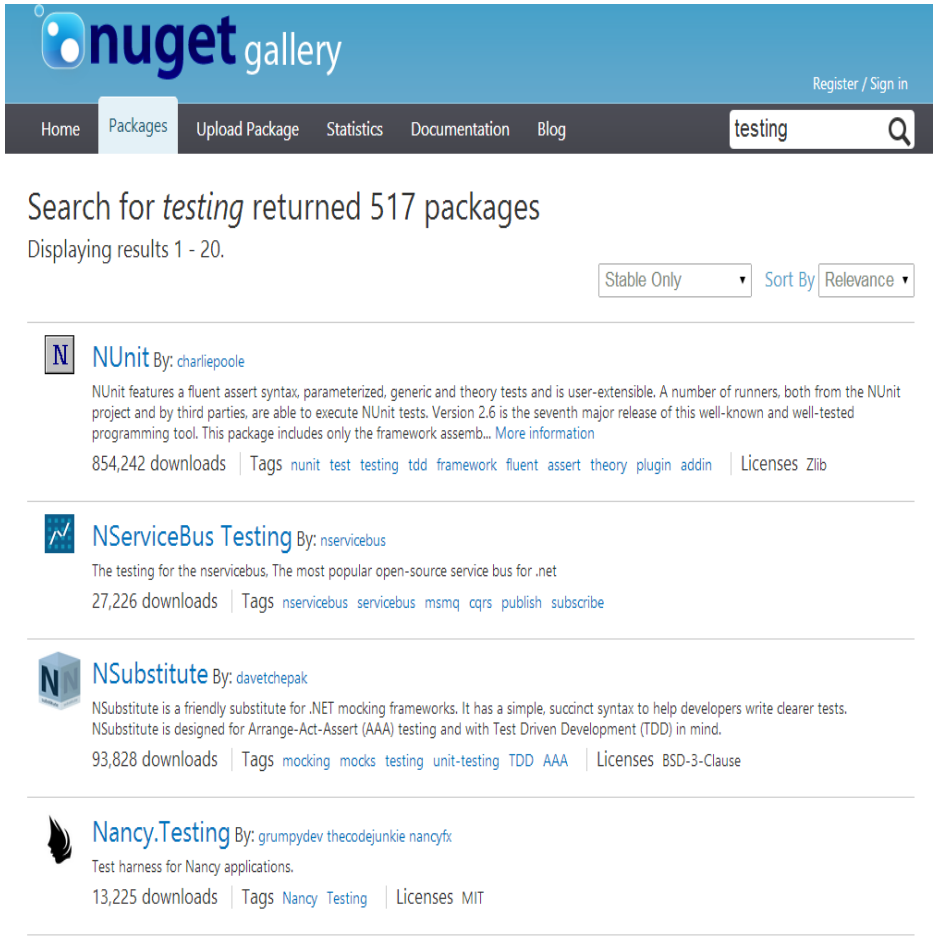
Use Nuget to Get Other Testing Frameworks

Search NuGet and install other testing frameworks like NUnit or XUnit



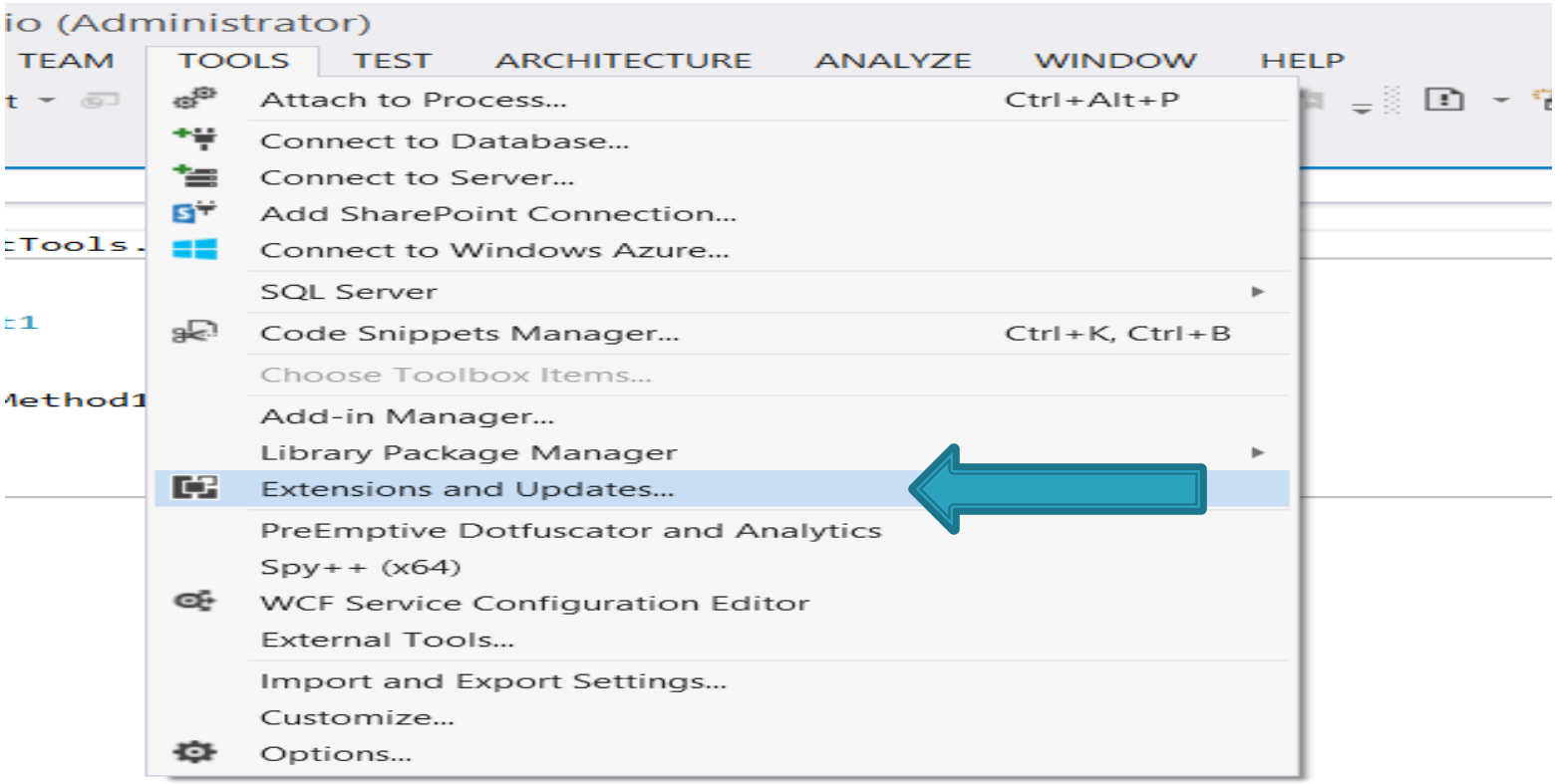


What is NuGet?



- NuGet is the package manager for the Microsoft development platform including .NET. The NuGet client tools provide the ability to produce and consume packages. The NuGet Gallery is the central package repository used by all package authors and consumers.

How to Get to NuGet?



Go to Tools → Extensions and Updates

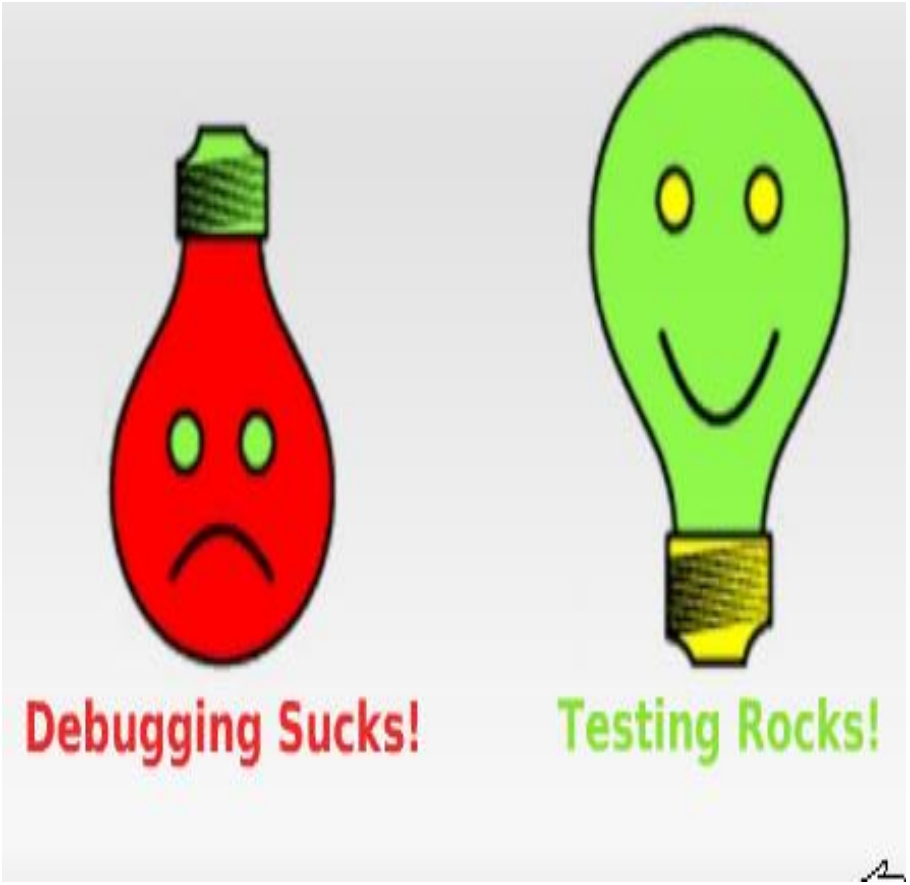
Disadvantages of TDD

- Learning curve
- Might add complexity to the code
- Design changes
- Continuous tweaking

Advantages of the Test Driven Development

- Higher quality code and fewer defects
- Well documented code
- Automated regression testing

Takeaways



- This picture is posted above the toilets at Google
- Its better to test and fix bugs in the code before it goes live
- Developers should strive to be proactive rather than reactive, remember the main goal of the IT shop is to ship working software. Good practices that will enhance productivity will need to be encouraged.

http://blog.felho.hu/wp-content/debuggingsuckststingrocks.png


TestDrivenDevelopment,
RedGreenRefractorCycle,
NuGet, LongRunningTests,
Fakes, Debugging,
VisualStudio2013, NUnit,
Shims, TestSetup,
TestExplorer,
XUnit Stubs,
UnitTests, CodeCoverage,
CodeTestFirst,


demo

Contact Info

- Twitter: https://twitter.com/bkraider
- Facebook: https://www.facebook.com/bkraider
- Linked In: http://www.linkedin.com/pub/bhanu-korremula/9/38/966
- Blog: http://bhanukorremula.wordpress.com/
- Work Email: korremulab@rcgov.us
- Personal Email: bkraider@gmail.com

References

- http://en.wikipedia.org/wiki/Test_Driven_Development
- http://msdn.microsoft.com/en-us/library/hh212233.aspx
- http://www.jrothman.com/2000/10/what-does-it-cost-you-to-fix-a-defect-and-why-should-you-care/
- http://superwebdeveloper.com/2009/11/25/the-incredible-rate-of-diminishing-returns-of-fixing-software-bugs/
- http://en.wikipedia.org/wiki/Unit_testing
- Pluralsight Test First Development, Visual Studio ALM
- http://www.amazon.co.uk/Engineering-Economics-Prentice-Hall-computing-technology/dp/0138221227
- http://sqa.stackexchange.com/questions/2899/industry-average-bug-fix-cost
- http://msdn.microsoft.com/en-us/library/hh270865.aspx
- http://www.microsoftvirtualacademy.com/training-courses/software-testing-with-visual-studio-2012-exam-70-497-jump-start#?fbid=HdC7BLz8mVC
- http://channel9.msdn.com/Series/Visual-Studio-2012-Premium-and-Ultimate-Overview/Visual-Studio-Ultimate-2012-Improving-quality-with-unit-tests-and-fakes
- http://academictips.org/blogs/the-story-of-a-woodcutter/
- http://www.flickr.com/photos/dullhunk/3712840085/
- http://blog.felho.hu/wp-content/debuggingsuckststingrocks.png
- http://www.nuget.org/
- http://www.nunit.org/
- http://xunit.codeplex.com/

References

- http://msdn.microsoft.com/en-us/library/dd537628.aspx
- http://stackoverflow.com/questions/64333/disadvantages-of-test-driven-development
- http://msdn.microsoft.com/en-us/library/ms182527.aspx
- http://msdn.microsoft.com/en-us/library/microsoft.visualstudio.testtools.unittesting.assert.aspx