

Unit Testing in Visual Studio 2010





ACCENTIENT EDUCATION SERIES

Committed to training success

+1 (877) 710-0841 ■ www.accentient.com

Unit Testing in Visual Studio 2010

Course Number:	UTVS
Version:	1.0
For software version:	2010

Copyright © 2011 Accentient, Inc. All rights reserved.

No part of this manual may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording, for any purpose without the express written permission of Accentient, Inc.

Images and Artwork

Images and artwork in this book were obtained through Flickr and are licensed under a Creative Commons 3.0 license. Visit <http://shrinkster.com/1buy> for more information.

All trademarks referenced are the property of their respective owners

Disclaimer

While Accentient takes great care to ensure the accuracy and quality of these materials, all material is provided without any warranty whatsoever, including, but not limited to, the implied warranties of merchantability or fitness for a particular purpose.

Unit Testing in Visual Studio 2010

Module 1 Unit Testing in .NET

Topics

- The role of the developer
- Unit tests explained
- .NET unit testing frameworks
- MSTest.exe
- Writing unit tests
- Lab

The “Developer”

- Developers deliver business value in the form of working software
 - This is done according to requirements, budget, and time
 - This is done according to a definition of done
 - This is done with good ethics
- Developers don’t just “write code”
 - They assist management with costs and estimates
 - They assist architects with the design
 - They write and run unit tests

Developers and Testers Overlap

- Every team and project is different
 - Some have developers who don’t test, with no testers
 - Some have developers who test, with no testers
 - Some have developers who don’t test, but have testers
 - Some have developers who test, and have testers
 - Some dedicated testers also code

Why Should a Developer Test Code

- **Developers are only human**
 - Humans make mistakes, have bad days, and don't always understand the requirements
- **Why the developer and not a "tester"?**
 - Who better to write the test than the person who is writing the code that it tests?
 - It is being a professional vs. being a cowboy, or being elite
- **Unit testing is probably the single most important quality practice a developer can adopt**

What is a Unit Test

- **A unit test is a fast, in-memory, consistent, automated, and repeatable test of a functional unit-of-work in the system**

More Technically ...

- Unit tests are used to exercise other source code by directly calling the methods of a class, passing appropriate parameters
 - If you include *Assert* statements, they can test the values that are produced against expected values
- Unit test methods reside in test classes, which are stored in source code files
 - Typically, the test class and all test methods are identified by using attributes

Why Do We Write Unit Tests?

- Safety net
- Reduce bugs
- Create better designs
- Enable meaningful refactoring
- Keep focus on getting to done
- Help document intent of code
- Force developer to think about goals
- Instant gratification
- Verify the existence of a bug

Unit Tests are Requirements

- Requirements should be clear and testable
- A well-understood requirement can be expressed
 - As a single unit test
 - As a single unit test driven by a data source
 - As a collection of unit tests
- When all related unit tests pass, then the requirement has been met
 - This is the basis for *Acceptance Test-Driven Development*

Unit Tests Test a Single Unit of Work

- A unit of work is any functional scenario in the system that contains logic
 - Typically this is just a single method
 - You may have multiple tests that test that method
- Tests that span multiple methods are referred to as *integration tests*

Unit Tests ...

- Do not adversely impact the environment
 - If a row in a database is added during the unit test, it should be removed in the cleanup code
- Are independent of other unit tests
 - *DeleteInvoiceTest* should be able to be run before (or after) the *AddInvoiceTest* which means setup/cleanup code
- Should be automated
 - Run with a single-click of a button
 - Run during Continuous Integration builds
- Can be run by anyone on the team

Discussion: What do we mean by a “unit”?

- Which of these is a true “unit”?



1

```
public int Add(int a, int b)
{
    return a + b;
}
```

4

```
public int Add(int a, int b)
{
    return Scrub(a + b);
    // Scrub() is a private
    // method that you wrote
}
```

2

```
public int Add(int a, int b)
{
    return Scrub(a + b);
    // Scrub() is a public
    // method that you wrote
}
```

5

```
public int Add(int a, int b)
{
    return Scrub(a + b);
    // Scrub() is part of the
    // .NET Framework
}
```

3

```
public int Add(int a, int b)
{
    return Scrub(a + b);
    // Scrub() calls a Web
    // Service in Munich
}
```

6

```
public int Add(int a, int b)
{
    return Scrub(a + b);
    // Scrub() looks up the
    // value in a database
}
```

Popular .NET Unit Testing Frameworks

- **MSTest**
 - Microsoft's unit (and other) test execution engine that is integrated with Visual Studio 2010
- **NUnit**
 - The original xUnit for .NET, ported from JUnit
 - <http://www.nunit.org>
- **xUnit.net**
 - A new, and increasingly more-popular unit testing tool written by the inventor of NUnit
 - <http://xunit.codeplex.com>
- **MbUnit**
 - A popular, non xUnit framework based on the Gallio Test Automation Platform
 - <http://www.mbunit.com>



xUnit

- **Various code-driven testing frameworks have come to be known collectively as xUnit**
 - xUnit is the closest thing to a unit test framework standard
 - Such frameworks are based on a design by Kent Beck, originally implemented for Smalltalk as SUnit
- **These frameworks allow testing of different elements (units) of software, such as functions and classes**
 - Erich Gamma and Kent Beck ported SUnit to Java (JUnit)
 - Framework was ported to C++ (CppUnit) and .NET (NUnit)
- **These frameworks are usually free and open source**
 - For a complete list: <http://bit.ly/MI6xa>

MSTest.exe

- MSTest.exe is Microsoft's command-line utility for executing tests
 - It is not Visual Studio's unit testing framework
 - It is not *Microsoft.VisualStudio.TestTools.UnitTesting*
- You can also use MSTest.exe to:
 - View test results
 - Save test results to disk
 - Publish test results to Team Foundation Server

Using MSTest.exe

- Use MSTest.exe to run tests from a command line
 - You can also view the test results from these test runs, save the results to disk, and publish to TFS
 - The executable can be found here:
<C:\Program Files\Microsoft Visual Studio 10.0\Common7\IDE>
- **Examples:**
 - `MSTest /testmetadata:worldcup.vsmDI /testlist:bvt`
 - `MSTest /testcontainer:wctests\bin\debug\wctests.dll`

Anatomy of a Unit Test

- A public class and method
 - Appropriate attributes on each
- Some setup code
- Execution of the target code
- Comparison between the expected and actual values

The 3A Pattern

- Writing a unit test can be done by following the 3 A's.
 - Arrange: setup the test
 - Act: exercise the code under test
 - Assert: verify the results

```
[TestMethod]
public void AddItemToCollection()
{
    var collection = new Collection<string>(); ⇐ Arrange
    string item = "foo";

    collection.Add(item);                    ⇐ Act

    Assert.AreEqual(item, collection[0]);    ⇐ Assert
}
```

Common Unit Test Attributes

Attribute	Description
[TestClass()]	Used to identify classes that contain test methods
[TestMethod()]	Used to identify test methods
[AssemblyInitialize]	Called before all tests in the assembly have run to allocate resources obtained by the assembly
[AssemblyCleanup]	Called after all tests in the assembly have run to free resources obtained by the assembly
[ClassInitialize()]	Called before any of the tests in the test class have run to allocate resources to be used by the test class
[ClassCleanup()]	Called after all the tests in the test class have run to free resources obtained by the test class
[TestInitialize()]	Called before the test to allocate and configure resources needed by all tests in the test class
[TestCleanup()]	Called after the test has run to free resources obtained by all the tests in the test class
[Timeout()]	Used to specify the time-out period of a unit test.

TestContext

- When you run a unit test, you are automatically provided with a TestContext instance
 - Your test class must define a TestContext property

TestContext Automatically Gives You...

- The name of the currently running unit test
- The test deployment directory
- The names of log files
- For DB info for data-driven testing
- The current test environment

Example of a Unit Test

```
[TestMethod()]
public void Paycheck_52000_Salary_Pays_2000_Test()
{
    // Bi-weekly

    string empName = "Mike Oobachesky";
    Employee target = new Employee(empName);
    target.YearlySalary = 52000;
    double actual = target.GetPaycheckAmount();
    double expected = 2000;
    Assert.AreEqual(expected, actual, "The
        employee was not paid the correct
        amount");
}
```

How Does a Unit Test Pass or Fail?

- Unit Tests are assumed to be in a passing state when they begin execution
- There are various ways a unit test can pass:
 - An Assert statement passes: `Assert.AreEqual(2,2);`
 - An expected exception was thrown
 - Unit test contains no code
- There are various ways a unit test can fail:
 - An Assert statement fails
 - An unexpected exception was thrown

Unit Test Assertions

- An assertion reports the final state of a unit test
- The Assert class has several important static methods
 - `AreEqual()` - tests whether two values are equal
 - `AreSame()` - tests whether two references are the same
 - `Fail()` - Causes the assertion to immediately fail
 - `Inconclusive()` - reports that the test could not be proven true or false and is the default for auto-created unit tests
 - `IsNull()` - used to determine whether an object is null
 - `IsInstanceOfType()` - used to determine if an object is a particular type
 - `IsTrue()` - used to test whether a statement is true
 - `AreNotEqual()`, `AreNotSame()`, `IsFalse()`, `IsNotNull()`, and `IsNotInstanceOfType()` also exist

Expected Exceptions

- Sometimes your code throws exceptions
 - This may be normal and needs to be tested
- Unit tests can verify that an expected exception occurs by using the `[ExpectedException()]` attribute

```
[TestMethod()]
[ExpectedException(typeof(System.DivideByZeroException))]
public void DivisionTest()
{
    MathClass target = new MathClass();
    int numerator = 4;
    int denominator = 0;
    int actual = target.Divide(numerator, denominator);
}
```

Test Class Inheritance

- Starting in Visual Studio 2010, test classes can now inherit members from other test classes
 - This means that you can create reusable tests in base test classes
 - Derived test classes can inherit tests from base test classes
 - This eliminates duplicated test code and gives developers more flexibility while unit-testing production code
- **Note:** The Visual Studio Test View window and Test List Editor display methods in base and child classes

Test Class Inheritance: Example

- Your production code has two implementations: SQL Server and Oracle
- Since both implementations must function in the same way, you create an abstract test class named *DBTestSuite* which contains standard test code and then create two sub-classes:
 - *SQLTestSuite* which implements the abstract Connect method in a way specific to SQL Server
 - *OracleTestSuite* which implements the abstract Connect method in a way specific to Oracle

Testing Private, Internal, and Friend Methods

- Unit tests can test private methods
- This can be coded manually, but it requires you to understand the intricacies of reflection
- It is easier to have Visual Studio 2010 generate the added infrastructure for you
 - Visual Studio will create a private accessor, which is an assembly through which the test can access a private method from outside the class of that method

Discussion: *Testing private methods*

- What do you think?
 - Is it a good idea to test a private method?
 - Shouldn't testing the public methods (that in-turn call the private method) be adequate?
 - Isn't one of the reasons to hide implementation is that it can change and wouldn't that break your tests?
 - If the private method is worth testing, shouldn't it be in its own class?
 - Can't you effectively test a private method by writing more tests against the public method that calls it?

Summary

- Unit testing is a design and development activity, not a testing activity
 - The developer who wrote the code should write the test
- Test at the unit level
- MSTest is the natural fit for Visual Studio developers
- You can test for expected exceptions
- Unit testing is probably the single most important quality practice a developer can adopt!

Lab

In this lab you will setup the learning environment and get familiar with the basics of writing and running a unit test.

- Install courseware
- Create and run a unit test
- Create and run an integration test
- Migrate NUnit tests to MSTest (optional)

AccentientTM



Lab 1: Unit Testing in .NET

Unit Testing in Visual Studio 2010

Estimated time to complete this lab: 60 minutes

The primary goal of unit testing is to take the smallest piece of testable software in the application, isolate it from the remainder of the code, and determine whether it behaves exactly as you expect. Each unit is tested separately before integrating them into modules to test the interfaces between modules.

In this lab you will gain experience writing and running unit tests in various ways. You will also see the differences between an integration test and a unit test, they both are executed and generate results in a similar fashion.

EXERCISE 0 – EXPLORE VIRTUAL PC

TASK – EXPLORE VIRTUAL PC

In this task, you will start Microsoft Virtual PC and familiarize yourself with the settings of the Virtual Machine. You will then start the image and familiarize yourself with sending Ctrl + Alt + Delete to the guest operating system.

1. Log on to the physical computer (the host) using the ID and password provided to you.

What is the User Name/Password for the physical computer? _____

2. Launch Microsoft **Virtual PC**.

You may want to make a desktop or start menu shortcut to this program. Ask your instructor for help locating the program if necessary.

3. Verify the settings for the course image.

How much Memory is assigned to the image? _____

The VM requires a minimum of 1.5GB of ram (1536MB). Please verify that enough memory is assigned.

4. Start the VM.

It will take a few moments for Virtual PC to start your (guest) image. The screens being displayed will resemble a physical computer during boot-up. In order to maximize the **memory allocated to the course image, don't run any other applications or services on the host computer.**

5. When the log-on screen appears, use the shortcut for CTRL + ALT + DEL and then logon to Windows.

What is the shortcut for Ctrl + Alt + Del? _____

You will login in the next exercise. Ask your instructor for help if necessary.

EXERCISE 1 – INSTALL COURSEWARE

TASK – INSTALL COURSEWARE FILES

In this task you will ensure that the lab files and other software required by this course are correctly installed.

1. Ensure that you are logged on to Windows using **Student** for the user name and **password** for the password.
2. Ensure that the following folder has been created and does not contain any files:

C:\Course

3. On your host computer (the computer running the VM software), locate the file containing the courseware files.

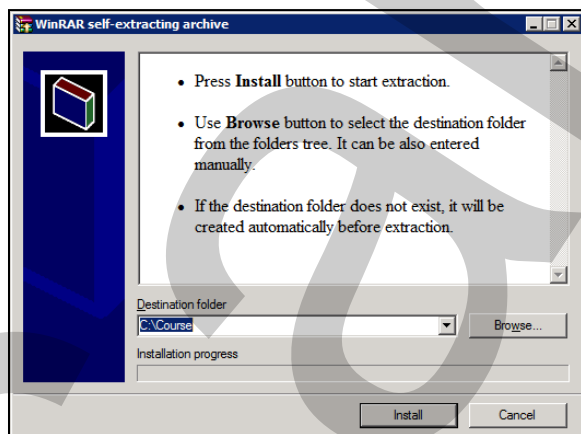
What is the full path? _____

Note: This file will be provided by your instructor. If you cannot locate this file, please email support@accentient.com to obtain a copy.

4. Drag the courseware file from the location above onto the desktop of your VM.

This step won't work for Windows 7 Virtual PC or Hyper-V. Ask your instructor for help.

5. In the VM, double-click the self-extracting courseware file to install the files.
6. Specify **C:\Course** as the **Destination** folder and click **Install**.



After the extraction of the files, you should see the following subfolders:

C:\Course\Labs
C:\Course\Software

EXERCISE 2 – CREATE AND RUN A UNIT TEST

TASK – EXPLORE COOLCALC.EXE

In this task, you will log on to Windows as Student, build and explore the calculator project that you will be testing.

1. Using **Windows Explorer**, navigate to **C:\Course\Labs\Lab01\CoolCalc**.
2. Double-click the **CoolCalc.sln** file.

This opens up a solution with two projects: CoolCalc, which is a C# console application and CoolCalcLib, which is a C# class library. Together they become a simple calculator application that you will use and test.

Note: If this is the first time you've run Visual Studio, then you may be asked to select a profile. Go ahead and select *General Development Settings* and wait for it to configure.

3. In the **Solution Explorer** window, expand the **CoolCalc** project.
4. Expand the **References** folder.

Do you see the reference to CoolCalcLib? _____

5. Double-click **Program.cs**.

This is a simple console application that takes input from the command line, assembles the calculation, executes it, and returns the value.

What happens when bad command line arguments are passed? _____

6. Close the code editor window.
7. In the **Solution Explorer** window, expand the **CoolCalcLib** project.

This project provides the implementation for the calculator itself and the operations it can perform. The operations are implemented using the Strategy pattern, whereby algorithms can be selected at runtime. The Strategy pattern defines a family of algorithms, encapsulates each one, and makes them interchangeable. You can learn more about this pattern here: <http://bit.ly/Hhms>.

Tip: Having the calculator implementation separated from the UI makes it possible to test the calculator logic directly. It also makes it possible to create multiple UIs (command line, desktop, Web, etc.) using the same implementation.

8. Double-click **Add.cs**.

This class implements the *IOperation* interface, which specifies 2 members:

- **Symbol**: the string a user types on the command line to invoke this class
- **PerformOperation**: the function called to perform the actual work.

The implementation in the Add class simply adds the 2 numbers passed in and returns the result.

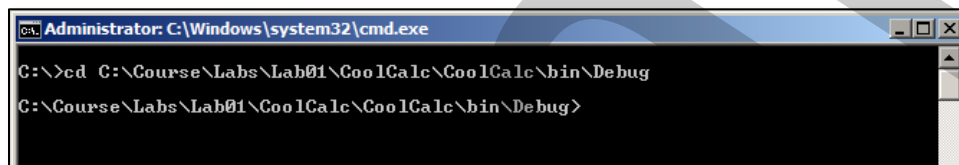
9. From the **Build** menu select **Rebuild Solution**.

10. Exit **Visual Studio**.

11. Open a command prompt window

You can use the Windows Key + R, type *cmd*, and press Enter as a shortcut, or you can use the command prompt that comes with Visual Studio 2010.

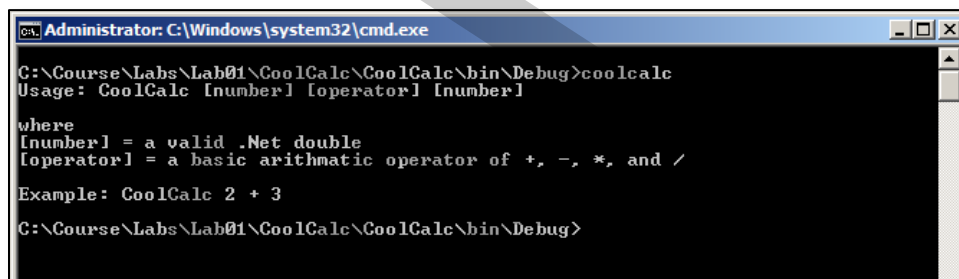
12. Navigate to **C:\Course\Labs\Lab01\CoolCalc\CoolCalc\bin\Debug**.



```
Administrator: C:\Windows\system32\cmd.exe
C:\>cd C:\Course\Labs\Lab01\CoolCalc\CoolCalc\bin\Debug
C:\Course\Labs\Lab01\CoolCalc\CoolCalc\bin\Debug>
```

13. Run the application by typing **coolcalc** and pressing **Enter**.

This will return the usage syntax:



```
Administrator: C:\Windows\system32\cmd.exe
C:\Course\Labs\Lab01\CoolCalc\CoolCalc\bin\Debug>coolcalc
Usage: CoolCalc [number] [operator] [number]
where
[number] = a valid .Net double
[operator] = a basic arithmetic operator of +, -, *, and /
Example: CoolCalc 2 + 3
C:\Course\Labs\Lab01\CoolCalc\CoolCalc\bin\Debug>
```

14. Enter **coolcalc.exe 2 + 3**.

This should return a 5. Note: you have to put a space before and after the "+" sign.

15. Press **Enter** to exit **CoolCalc**.

16. Close the command window.

TASK – REVIEW EXISTING UNIT TESTS

In this task, you will open and review a test project, seeing the unit tests that have already been created.

1. Using **Windows Explorer**, navigate to **C:\Course\Labs\Lab01\CoolCalcTests**.
2. Double-click the **CoolCalcTests.sln** file.

This opens up a solution that contains one project: CoolCalcTests which is a C# test project that contains one basic unit test file AddTests.cs.

3. Double-click **AddTests.cs**.

This public class contains a single public method. Both the class and the method are decorated with the appropriate attributes to indicate that the method is a test that can be executed.

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

namespace CoolCalcTests
{
    [TestClass]
    public class AddTests
    {
        [TestMethod]
        public void PerformOperationTest()
        {
            Add target = new Add();
            double arg1 = 2;
            double arg2 = 3;
            double expected = 5;
            double actual;
            actual = target.PerformOperation(arg1, arg2);
            Assert.AreEqual(expected, actual);
        }
    }
}
```

4. Close the code editor window.
5. From the **Build** menu select **Rebuild Solution**.
6. Exit **Visual Studio**.

TASK – RUN UNIT TESTS USING MSTEST.EXE

In this task, you will execute the PerformOperationTest using the MSTest.exe command line utility.

1. Using **Windows Explorer**, navigate to **C:\Course\Labs\Lab01**.
2. Right-click the **MSTest-Help.bat** batch file and select **Edit**.

This batch file will run mstest.exe from its folder, passing a question mark switch to return help on the utility.

Note: depending on the your Windows' settings, you may be prompted with a security warning message as you edit and run the batch files in this lab.

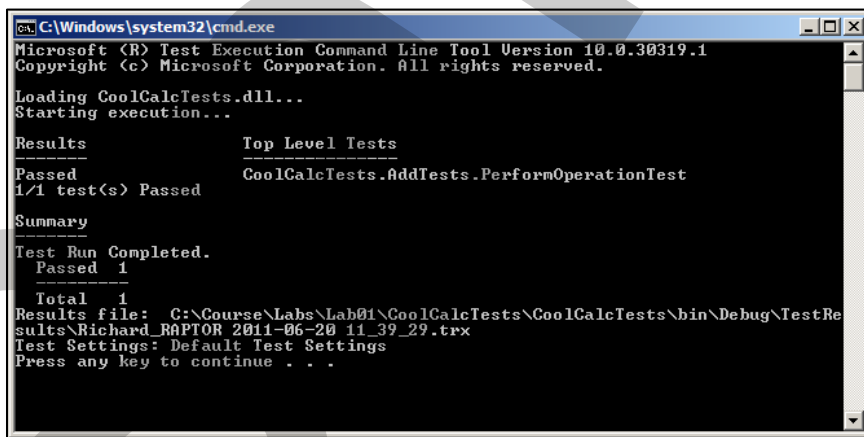
3. Close **Notepad**.
4. Double-click **MSTest-Help.bat**.

What version is your MSTest? _____

5. Press any key to close the command window.
6. Right-click the **MSTest-CoolCalcTests-PerformOperation.bat** batch file and select **Edit**.

This batch file will change the current folder to the one that contains the CoolCalcTests test assembly and then run mstest.exe from its folder, passing the testcontainer (the DLL) and the test (PerformOperationTest) to run.

7. Close **Notepad**.
8. Double-click **MSTest-CoolCalcTests-PerformOperation.bat**.



```
C:\Windows\system32\cmd.exe
Microsoft (R) Test Execution Command Line Tool Version 10.0.30319.1
Copyright (c) Microsoft Corporation. All rights reserved.
Loading CoolCalcTests.dll...
Starting execution...

Results          Top Level Tests
-----          -
Passed          CoolCalcTests.AddTests.PerformOperationTest
1/1 test(s) Passed

Summary
-----
Test Run Completed.
  Passed 1
  Total 1
Results file: C:\Course\Labs\Lab01\CoolCalcTests\CoolCalcTests\bin\Debug\TestRe
sults\Richard_RAPTOR_2011-06-20_11_39_29.trx
Test Settings: Default Test Settings
Press any key to continue . . .
```



Did your unit test pass or fail? _____

- Press any key to close the command window.
- Navigate to **C:\Course\Labs\Lab01\CoolCalcTests\CoolCalcTests\bin\Debug\TestResults**.

Directly under TestResults is one folder for each test run that has been started. The test run folder has the same name as the test run, as displayed in the Test Results and the Test Runs window in Visual Studio – which you will see in the next module. The default format for test run name is <user name>@<computer name> <date> <time>. The test run folder also contains the cleanup scripts (if any) that are run before and after the tests.

How many subfolders do you see here? _____

What is the prefix MSTest gave to your folder and log file? _____

Name ^	Date modified	Type	Size
 Richard_RAPTOR 2011-06-20 11_39_29	6/20/2011 11:39 AM	File folder	
 Richard_RAPTOR 2011-06-20 11_39_29.trx	6/20/2011 11:39 AM	Visual Studio Test Results File	3 KB

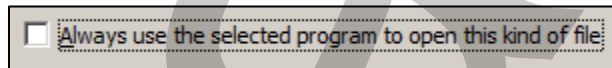
- Double-click the subfolder and navigate to the **Out** folder.

Each test run folder contains a folder named **Out**. The Out folder is the actual deployment folder. Assemblies and other files or folders that are required for the test run are copied to the <solution>\TestResults\<test run>\Out folder every time that a test run is started. Sometimes you will see an **In** folder as well.

How many files do you see listed here? _____

- Navigate back up to the **TestResults** folder.
- Right-click on the **.TRX** file and select **Open-with** and select **Internet Explorer**.

Note: be sure and clear the “Always use ...” checkbox:



Reviewing the file in Internet Explorer, what format is a .trx file? _____

You can specify the `/resultsfile:[file name]` option when running MSTest to specify the name of the file to which you want to save test results.

- Exit **Internet Explorer**.

TASK – ADD ADDITIONAL UNIT TESTS

In this task, you will add additional unit tests that you will run from the command line using MSTest.exe.

1. Using **Windows Explorer**, navigate to **C:\Course\Labs\Lab01\CoolCalcTests**.
2. Double-click the **CoolCalcTests.sln** file.
3. Double-click **AddTests.cs**.
4. Rename the **PerformOperationTest** method to **Add_2_3_Returns_5**.

```
[TestMethod]
public void Add_2_3_Returns_5()
{
    Add target = new Add();
    double arg1 = 2;
    double arg2 = 3;
    double expected = 5;
    double actual;
    actual = target.PerformOperation(arg1, arg2);
    Assert.AreEqual(expected, actual);
}
```

5. Copy the **Add_2_3_Returns_5** method (including the [TestMethod] attribute) and paste two copies below the original method.
6. Rename the first copy to **Add_3_2_Returns_5**.
7. Rename the second copy to **Add_2_2_Returns_22**.
8. Alter the **Add_3_2_Returns_5** method accordingly:

```
[TestMethod]
public void Add_3_2_Returns_5()
{
    Add target = new Add();
    double arg1 = 3;
    double arg2 = 2;
    double expected = 5;
    double actual;
    actual = target.PerformOperation(arg1, arg2);
    Assert.AreEqual(expected, actual);
}
```

- Alter the **Add_2_2_Returns_22** method accordingly:

```
[TestMethod]
public void Add_2_2_Returns_22()
{
    Add target = new Add();
    double arg1 = 2;
    double arg2 = 2;
    double expected = 22;
    double actual;
    actual = target.PerformOperation(arg1, arg2);
    Assert.AreEqual(expected, actual);
}
```

Yes, I know – this is a bad test. It would be accurate for string addition though!

- Save your changes.
- Close the code editor window.
- From the **Build** menu select **Rebuild Solution**.
- Exit **Visual Studio**.
- Using **Windows Explorer**, navigate to **C:\Course\Labs\Lab01**.
- Right-click the **MSTest-CoolCalcTests-All.bat** batch file and select **Edit**.

This batch file is similar to the first one, but runs all tests. Notice the use of the wildcard (*) character in the test to run.

- Close **Notepad**.
- Double-click **MSTest-CoolCalcTests-All.bat**.

Did all of your unit tests pass? _____

Which one(s) failed? _____

- Press any key to close the command window.
- Navigate to **C:\Course\Labs\Lab01\CoolCalcTests\CoolCalcTests\bin\Debug\TestResults**.
- Double-click on the most recent .trx file.

This will open the test results file in Visual Studio. You can see the results in the Test Results window. You will learn more about this window in the next module.

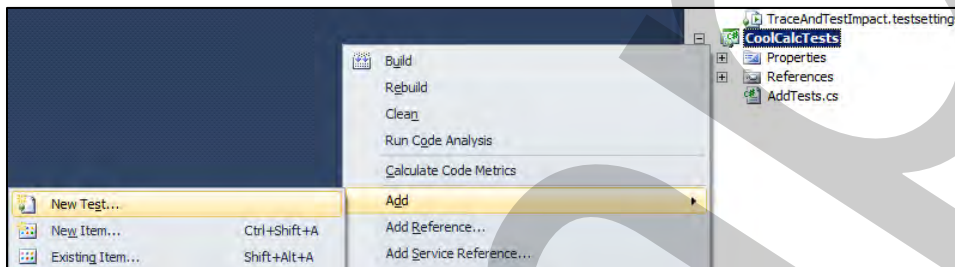
- Exit **Visual Studio**.

EXERCISE 3 – CREATE AND RUN AN INTEGRATION TEST

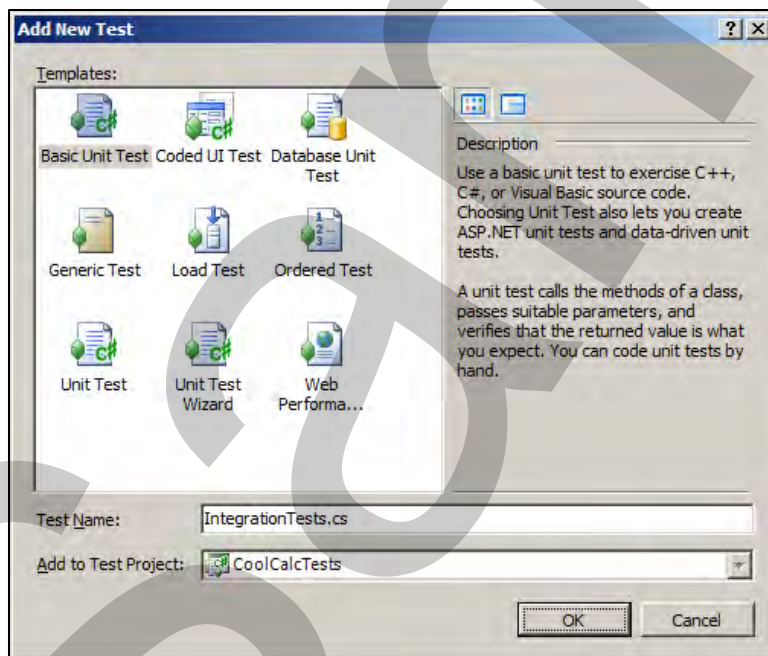
TASK – CREATE AND RUN AN INTEGRATION TEST

In this task, you will add another unit test, which is actually an integration test, to the test project, build it, and then execute it.

1. Using **Windows Explorer**, navigate to **C:\Course\Labs\Lab01\CoolCalcTests**.
2. Double-click the **CoolCalcTests.sln** file.
3. In **Solution Explorer**, right-click on the **CoolCalcTests** project and select **Add > New Test**.



4. Select **Basic Unit Test**, name it **IntegrationTests.cs**, and click **OK**.



5. Replace the contents of the IntegrationTests.cs with the contents of **C:\Course\Labs\Lab01\IntegrationTests.cs.txt**.

As you can see, this test is spinning up a lot of infrastructure, making calls using reflection, etc. It is way more than a unit test – which is ok, so long as you also have unit tests and you write them first.

6. Save your changes.
7. Close the code editor window.
8. From the **Build** menu select **Rebuild Solution**.
9. Exit **Visual Studio**.
10. Using **Windows Explorer**, navigate to **C:\Course\Labs\Lab01**.
11. Right-click the **MSTest-CoolCalcTests-Integration.bat** batch file and select **Edit**.

This batch file just runs the new integration test.

12. Close **Notepad**.
13. Double-click **MSTest- CoolCalcTests-Integration.bat**.

Did the integration test pass? _____

14. Press any key to close the command window.
15. Double-click **MSTest-CoolCalcTests-All.bat**.

Did this test run include the new integration test? _____

Integration tests appear just like any other unit test in the project. You can execute them by name or include them in the (*) wildcard.

16. Press any key to close the command window.

EXERCISE 4 – MIGRATE NUNIT UNIT TESTS TO MSTEST (OPTIONAL)

If you and your team want to experience the tight integration with MSTest and Visual Studio, you will need to abandon NUnit. This optional exercise briefly shows you the difference between writing and running unit tests in NUnit versus MSTest and then the steps you need to take to migrate from NUnit to MSTest.

TASK – INSTALL NUNIT

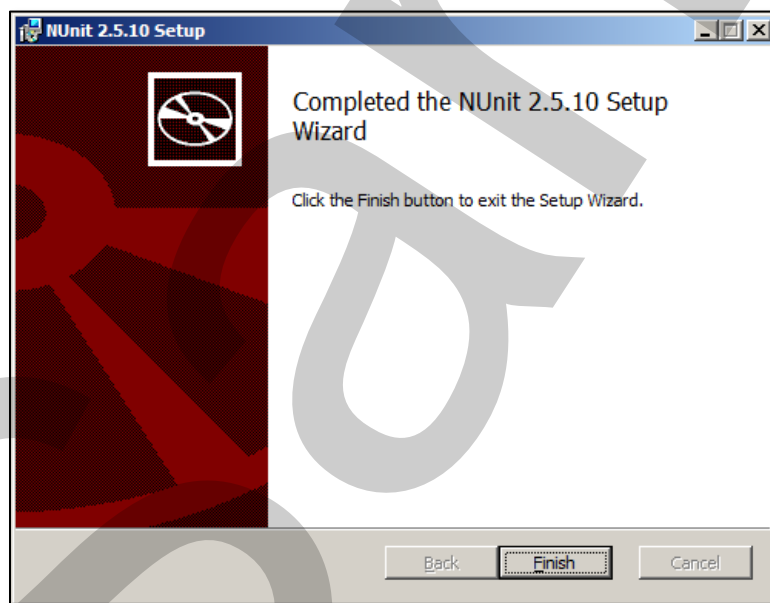
In this task, you will install NUnit.

1. Using **Windows Explorer**, navigate to **C:\Course\Software**.
2. Double-click **NUnit-x.x.xx.xxxxx.msi**.

If prompted with a security warning, click Run.

3. Click **Next**.
4. Accept the terms and click **Next**.
5. Click **Complete**.
6. Click **Install**.

Installation only takes a few moments.



7. Click **Finish**.

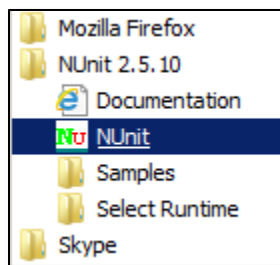
TASK – EXPLORE NUNIT UNIT TESTS

In this task, you will open a sample project that has been decorated with NUnit-specific attributes.

1. Using **Windows Explorer**, navigate to **C:\Course\Labs\Lab01\NUnit**.
2. Double-click the **CSharp.sln** file.

This sample solution demonstrates the use of NUnit and is organized as follows:

- **CS-Failures** project: contains failing and disabled unit tests
 - **CS-Money** project: our system under test – a money and moneybag implementation
 - **CS-Syntax** project: contains a variety of NUnit Assert examples
3. From the **Build** menu select **Rebuild Solution**.
 4. Exit **Visual Studio**.
 5. From the **Start** menu select **All Programs > NUnit x.x.xx > NUnit**.

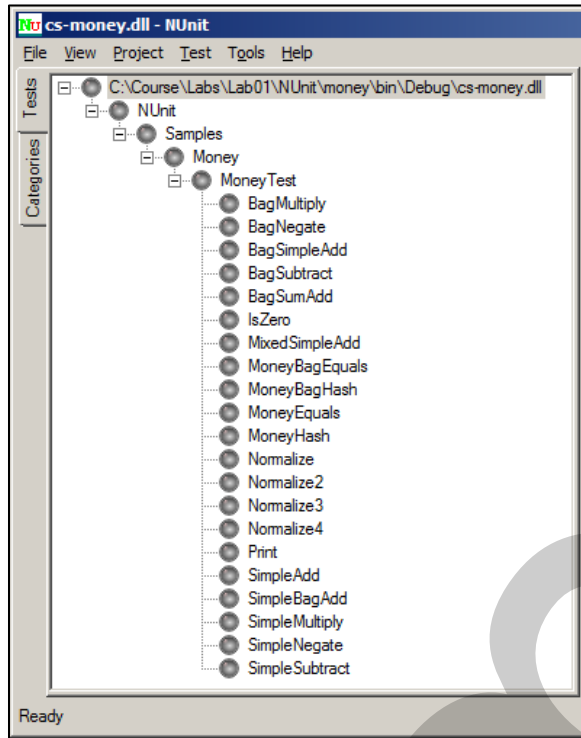


6. From the **NUnit** interface, select **File > Open Project**.
7. Navigate to **C:\Course\Labs\Lab01\NUnit\money\bin\Debug** and open the assembly **cs-money.dll**.

It takes a few moments for the assembly to open and display all of the tests.

- Expand all of the folder levels.

You should see a few folders and then several unit tests in this assembly:



- Highlight the root node in the tree and press **F5**.

Note: Pressing F5 runs all the tests whereas pressing F6 runs only the selected ones. In this case, pressing either function key has the same effect.

How many tests were run? How many passed? _____

Feel free to play around with the NUnit interface if you haven't before.

- From the **File** menu select **Close**.
- From the **NUnit** interface, select **File > Open Project**.
- Navigate to **C:\Course\Labs\Lab01\NUnit\failures\bin\Debug** and open the assembly **cs-failures.dll**.
- Press **F5** to run all of these tests.

As you can see, no tests passed and one of them was ignored.

- Exit **NUnit**.

TASK – MIGRATE NUNIT UNIT TESTS TO MSTEST

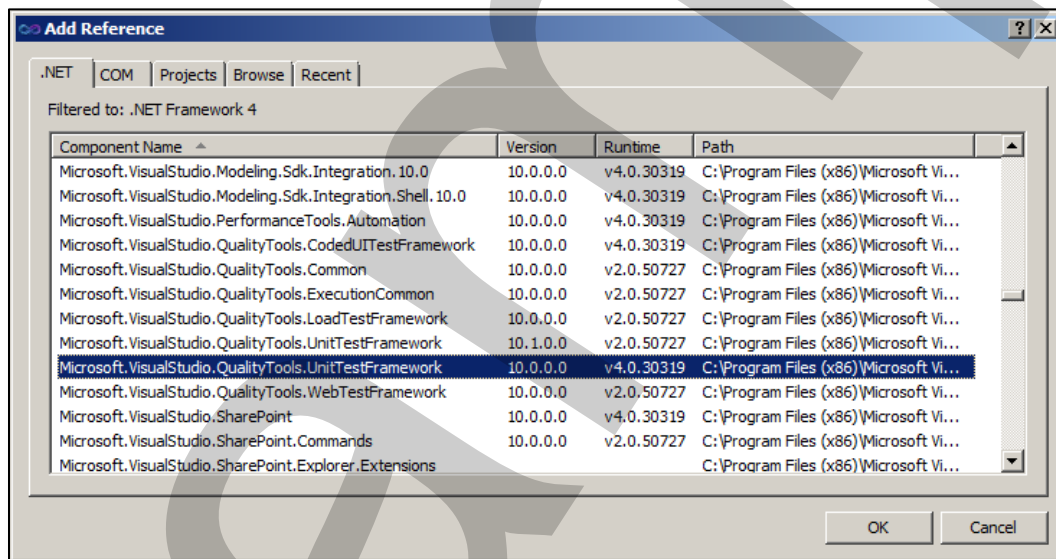
In this task, you will return to the NUnit sample solution and enable it to run within Visual Studio 2010.

1. Using **Windows Explorer**, navigate to **C:\Course\Labs\Lab01\NUnit**.
2. Double-click the **CSharp.sln** file.
3. In the **Solution Explorer** window, expand the **cs-money** project and double-click on the **MoneyTest.cs** file.
4. Review the code.

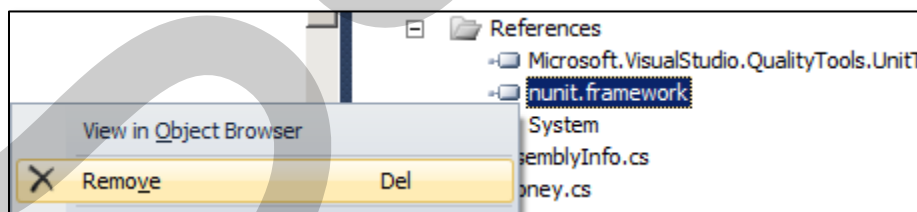
What attribute does NUnit use to decorate a test class? _____

What attribute does NUnit use to decorate a test method? _____

5. In **Solution Explorer**, right-click on the **cs-money** project and select **Add Reference**.
6. Select the **Microsoft.VisualStudio.QualityTools.UnitTestingFramework** (v4 version) and click **OK**.



7. In **Solution Explorer**, right-click **nunit.framework** and select **Remove**.



8. In the **MoneyTest.cs** file, replace this line:

```
using NUnit.Framework;
```

with this line:

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

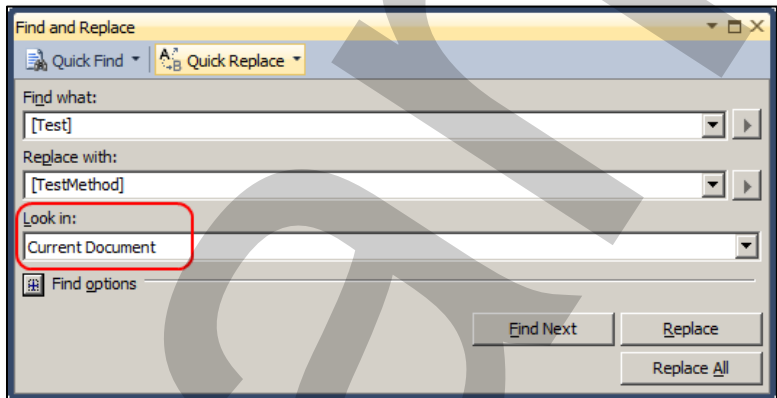
```
namespace NUnit.Samples.Money
{
    using System;
    using Microsoft.VisualStudio.TestTools.UnitTesting;
    /// <summary>
    ///
```

9. Using search and replace, change the following

- [TestFixture] to [TestClass]
- [Test] to [TestMethod]
- [Setup] to [TestInitialize]

There are other possible attributes that you may need to change. Please see this article for a full list: <http://bit.ly/IVILHV>.

Note: if you are going to use Visual Studio's Find and Replace (Ctrl + H), be sure to scope your changes to just the current document.



10. Locate the [TestInitialize] Setup method and mark it as public.

```
///
[TestInitialize]
public void Setup()
{
    f12CHF= new Money(12, "CHF");
}
```

11. Save your changes.
12. Close the code editor window.
13. In **Solution Explorer**, right-click on the **cs-money** project and select **Unload Project**.
14. Right-click on the **cs-money** project and select **Edit cs-money.csproj**.
15. Add a new element below the <ProjectType> element for the <ProjectTypeGuids>:

Note: you can find this text in C:\Course\Labs\Lab01\ProjectTypeGuids.txt

```
<?xml version="1.0" encoding="utf-8"?>
<Project ToolsVersion="4.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003" DefaultTargets="Build">
  <PropertyGroup>
    <ProjectType>Local</ProjectType>
    <ProjectTypeGuids>{3AC096D0-A1C2-E12C-1390-A8335801FDAB};{FAE04EC0-301F-11D3-BF4B-00C04F79EFBC}</ProjectTypeGuids>
    <ProductVersion>8.0.30319</ProductVersion>
    <SchemaVersion>2.0</SchemaVersion>
    <ProjectGuid>{11EDF872-A04D-4F75-A1BF-71168DC86AF3}</ProjectGuid>
    <Configuration Condition=" '$(Configuration)' == '' ">Debug</Configuration>
```

16. Save your changes.
17. In **Solution Explorer**, right-click on the **cs-money** project and select **Reload Project**.
18. From the **Build** menu select **Rebuild Solution**.

Note: You may get some warnings, which is ok. If you get any errors, however, please double-check the replacement work you did.

19. Exit **Visual Studio**.
20. Using **Windows Explorer**, navigate to **C:\Course\Labs\Lab01**.
21. Right-click the **MSTest-NUnit.bat** batch file and select **Edit**.

This batch file will run mstest.exe from its folder, passing a question mark switch to return help on the utility.

22. Close **Notepad**.

23. Double-click **MSTest-NUnit.bat**.

Did all of your unit tests pass? _____

```
C:\Windows\system32\cmd.exe
Microsoft (R) Test Execution Command Line Tool Version 10.0.30319.1
Copyright (c) Microsoft Corporation. All rights reserved.

Loading cs-money.dll...
Starting execution...

Results
-----
Top Level Tests
-----
Passed NUnit.Samples.Money.MoneyTest.BagMultiply
Passed NUnit.Samples.Money.MoneyTest.BagNegate
Passed NUnit.Samples.Money.MoneyTest.BagSimpleAdd
Passed NUnit.Samples.Money.MoneyTest.BagSubtract
Passed NUnit.Samples.Money.MoneyTest.BagSumAdd
Passed NUnit.Samples.Money.MoneyTest.IsZero
Passed NUnit.Samples.Money.MoneyTest.MixedSimpleAdd
Passed NUnit.Samples.Money.MoneyTest.MoneyBagEquals
Passed NUnit.Samples.Money.MoneyTest.MoneyBagHash
Passed NUnit.Samples.Money.MoneyTest.MoneyEquals
Passed NUnit.Samples.Money.MoneyTest.MoneyHash
Passed NUnit.Samples.Money.MoneyTest.Normalize
Passed NUnit.Samples.Money.MoneyTest.Normalize2
Passed NUnit.Samples.Money.MoneyTest.Normalize3
Passed NUnit.Samples.Money.MoneyTest.Normalize4
Passed NUnit.Samples.Money.MoneyTest.Print
Passed NUnit.Samples.Money.MoneyTest.SimpleAdd
Passed NUnit.Samples.Money.MoneyTest.SimpleBagAdd
Passed NUnit.Samples.Money.MoneyTest.SimpleMultiply
Passed NUnit.Samples.Money.MoneyTest.SimpleNegate
Passed NUnit.Samples.Money.MoneyTest.SimpleSubtract
21/21 test(s) Passed

Summary
-----
Test Run Completed.
  Passed 21
-----
  Total 21
Results file: C:\Course\Labs\Lab01\NUnit\money\bin\Debug\TestResults\Richard_RA
PTOR 2011-06-20 14_58_53.trx
Test Settings: Default Test Settings
Press any key to continue . . . _
```

24. Press any key to close the command window.

Summary

You can use the MSTest.exe program to run tests from a command line. MSTest.exe automatically displays a run summary to the command prompt window as well as generating a test results file in an **XML format**. **You can tweak many of MSTest's behaviors** as well.

In the next lab you will do all test-related work from within Visual Studio, not just the creating and managing of the tests.