Course Goals and Non Goals

- Course Goals
- To introduce various developer tools for Unit Testing, Code Coverage, Code Analysis and Logging

Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

 2

Pre-requisites

- C# 5.0
- ASP.NET 4.5
- ADO.NET 4.5

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 3

Day Wise Schedule

- Day 1
  - Lesson 1: Working with NUnit & NCover
  - Lesson 2: Using FxCOP, StyleCOP & Log4Net

Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

 4

Table of Contents

Table of Contents

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 6

References

- Student Guide
  - PPT along with notes
- Lab Book
  - Lab Book consists of Walkthroughs to guide the participants throughout and <<To Do>> Exercises to test their knowledge and skills

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 7

Developer Tools for .NET

Lesson 2: Overview of FxCOP, StyleCop & Log4NET

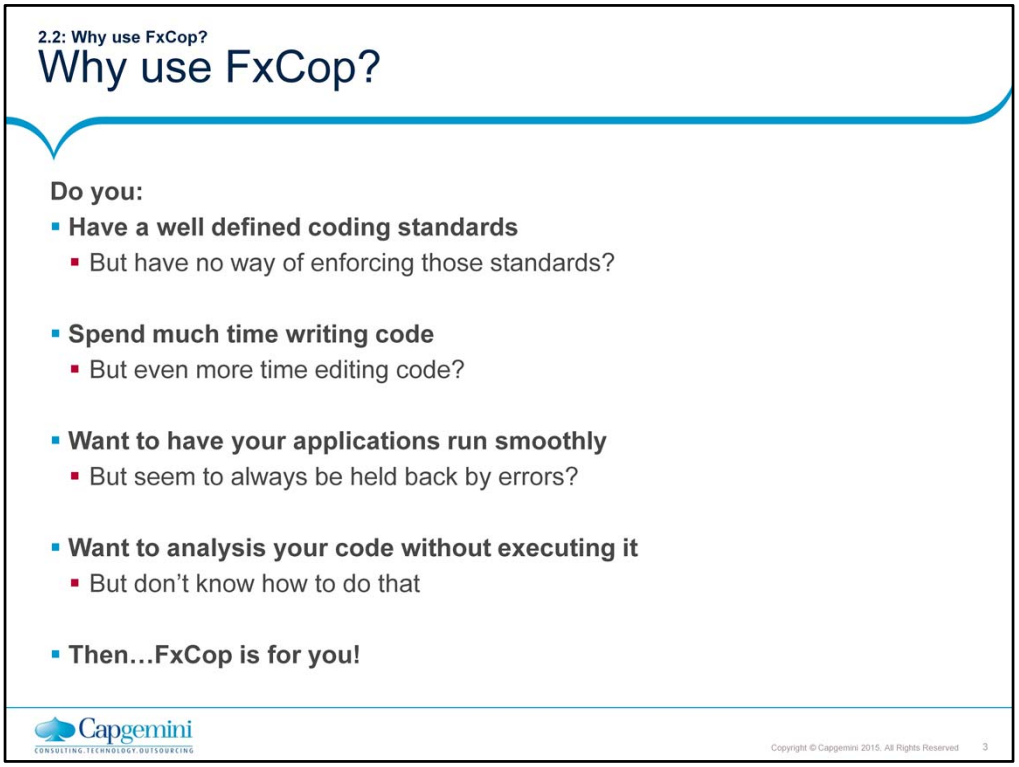
2.1: Static Code Analysis

Static Code Analysis

- From Wikipedia: “The analysis of computer software that is performed without actually executing programs built from that software”
  - In most cases the analysis is performed on some version of the source code and in the other cases some form of the object code
- In .NET, this means analyzing managed compiled code (IL)

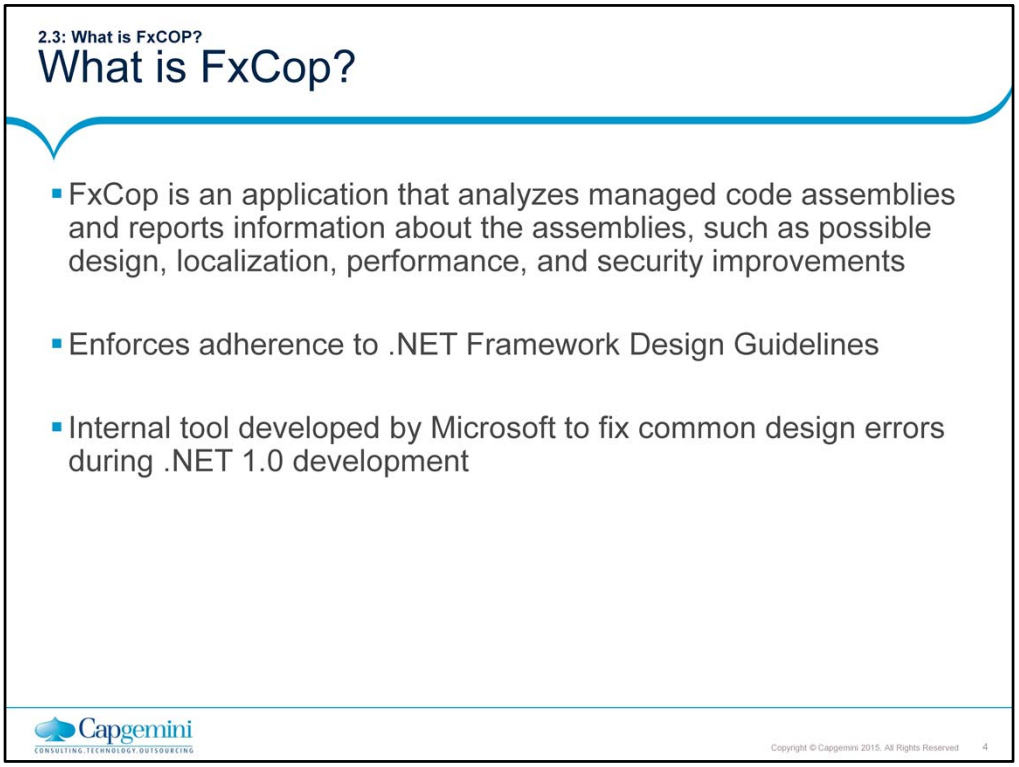
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 2



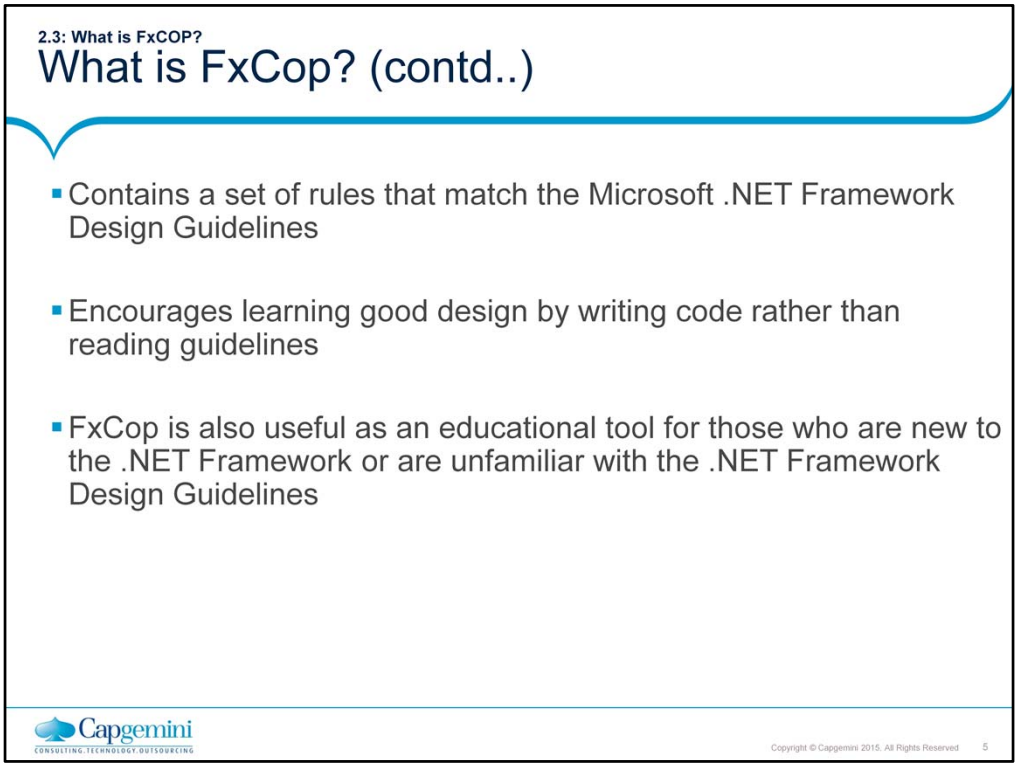
The nuances of writing code present many problems, and perhaps the most annoying are those that involve the tendency for small errors to plague your code.

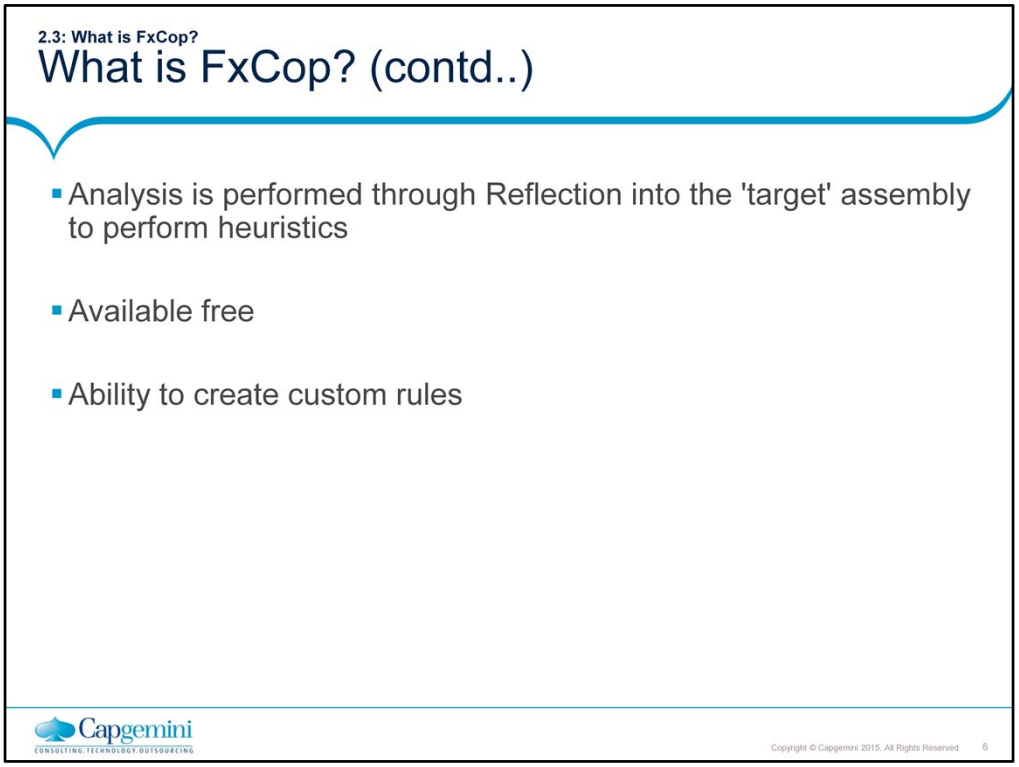
FxCop is designed to solved these problems.



Many of the issues concern violations of the programming and design rules set forth in the Microsoft .NET Framework Design Guidelines, which are Microsoft's guidelines for writing robust and easily maintainable code using the .NET Framework.

FxCop is intended for class library developers; however, anyone creating applications that should conform to .NET Framework best practices will benefit.


2.3: What is FxCOP?
What is FxCop? (contd..)
Contains a set of rules that match the Microsoft .NET Framework Design Guidelines
Encourages learning good design by writing code rather than reading guidelines
FxCop is also useful as an educational tool for those who are new to the .NET Framework or are unfamiliar with the .NET Framework Design Guidelines
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved
5



Microsoft has put a lot of effort into disseminating best practices and guidelines for writing code to be used for the .NET framework. The FxCop project began as an internal one, aimed at ensuring that Microsoft developers followed their own rules.

FxCop is now available from http://www.gotdotnet.com/team/fxcop.

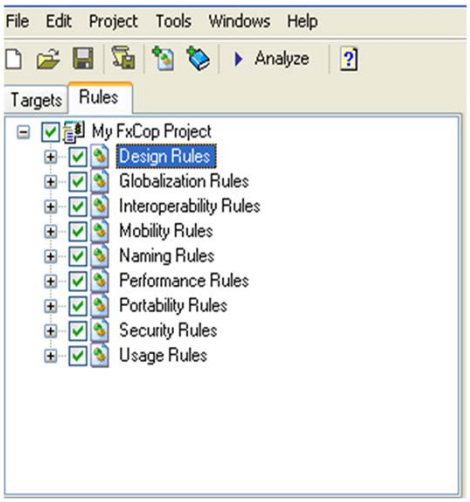
Introspection is the new element of FxCop, contained in version 1.3. The old Reflection version required FxCop to shutdown to complete a fix, then run again to recompile. Now, since the assemblies are not locked, there is no need to shut down. Those who have used the old Reflection engine will be happy to hear that rules can be updated by just changing a few base class names.

The rules for FxCop include everything from ensuring that you use COM interop correctly, to ensuring proper globalization, to enforcing rules for writing high-performance code

2.4: FxCop Rules

FxCop Rules

- It Contains over 200 rules in the following areas:
  - Library design
  - Localization
  - Naming conventions
  - Performance
  - Security
- Defined in logical groups
- Extensible – you can write
- your own rules!



Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 7

2.4: FxCop Rules

FxCop Rules (contd..)

- FxCop tests rules against assemblies and reports on failed rules
  - FxCop can be applied to any .NET language because it works on assemblies and not code
  - The rules included with FxCop are based upon “Microsoft .NET Framework Design Guidelines”

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 8

2.5: FxCOP Tool

FxCOP Tool

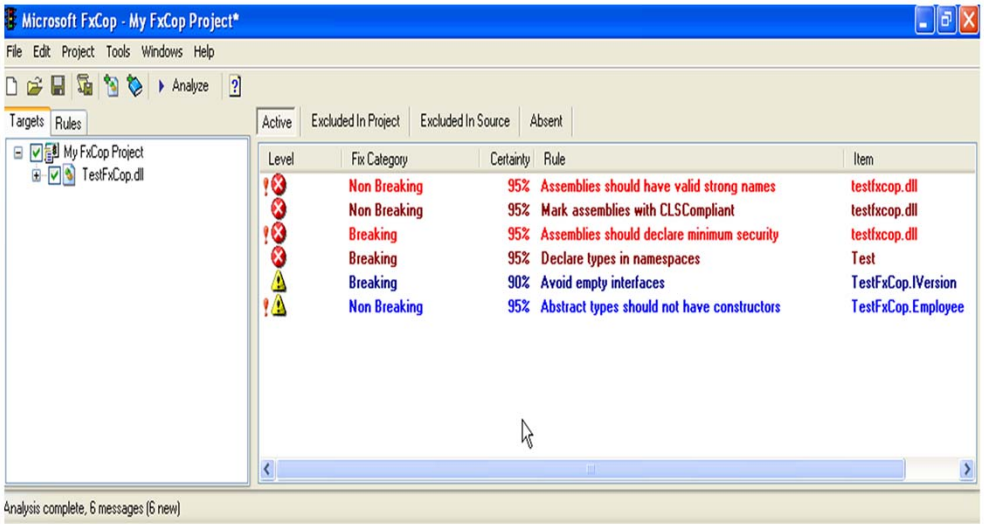
- FxCop is designed to be fully integrated into the software development cycle
- It is distributed as a fully featured application with a graphical user interface (FxCop.exe) for interactive work
- Also available as a command-line tool (FxCopCmd.exe) suitable for use as part of automated build processes
- It can be integrated with Microsoft Visual Studio .NET as an external tool

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 9

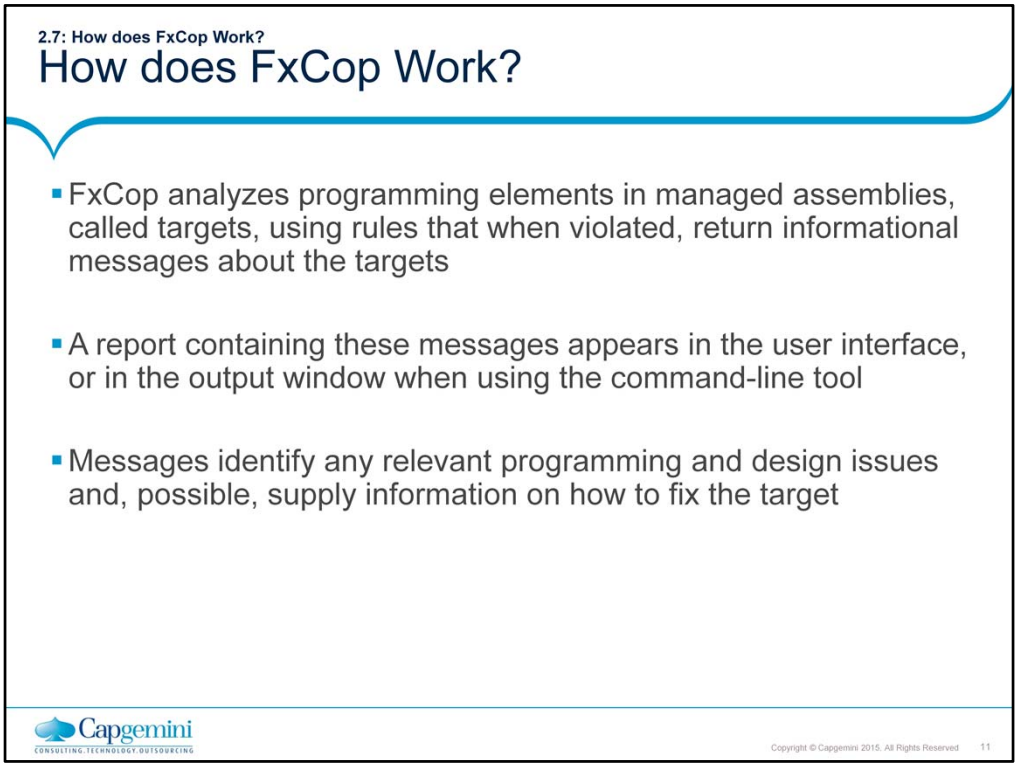
2.6: FxCop in Action

FxCop in Action



Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 10



FxCop analyzes programming elements in managed assemblies, called targets, and provides an informational report containing messages about the targets, including suggestions on how to improve the source code used to generate them. FxCop represents the checks it performs during an analysis as rules. A rule is managed code that can analyze targets and return a message about its findings. Rule messages identify any relevant programming and design issues and, when possible, supply information on how to fix the target.

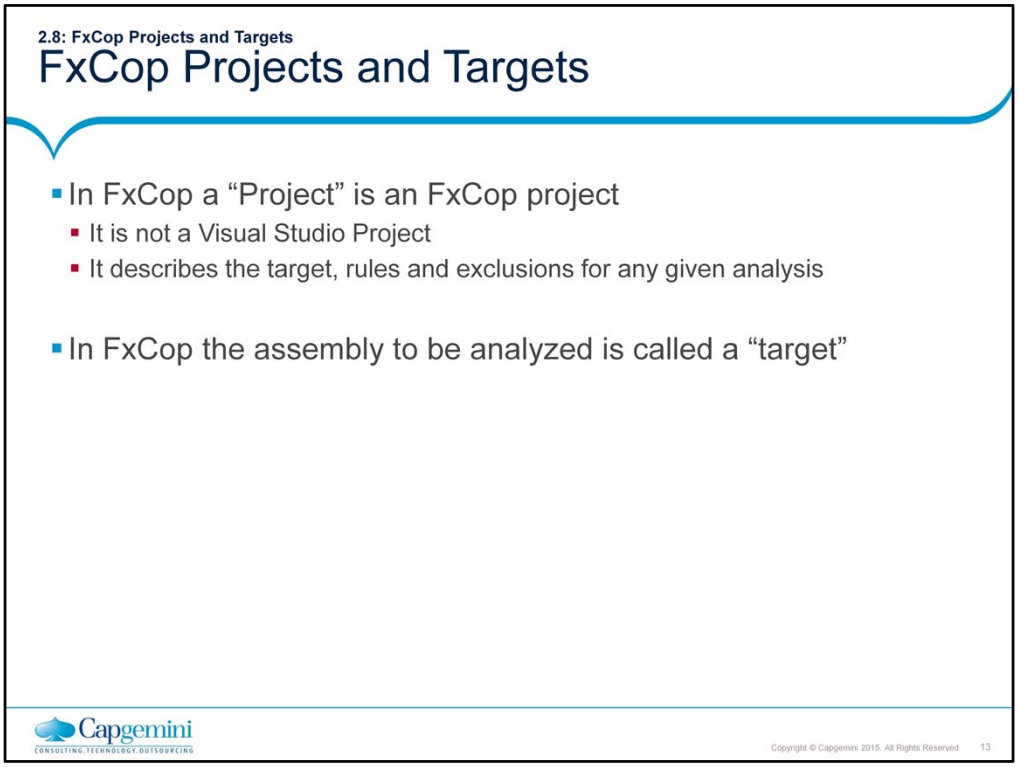
2.7: How does FxCop Work?

How does FxCop Work? (contd..)

- A message is associated with a specific rule and a specific target and can be excluded from further analysis
- A default set of rules is provided and additional custom rules can be created using the FxCop SDK

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 12



Projects are used to specify the set of assemblies to be analyzed, the rules used to analyze the assemblies, the excluded messages, and settings to customize the saved project or report file.

2.9: Using FxCop

Using FxCop

- Using FxCop, you can:
  - Control which rules are applied to targets
  - Exclude a rule message from future reports
  - Apply style sheets to FxCop reports
  - Filter and save messages
  - Save and reuse application settings in FxCop projects

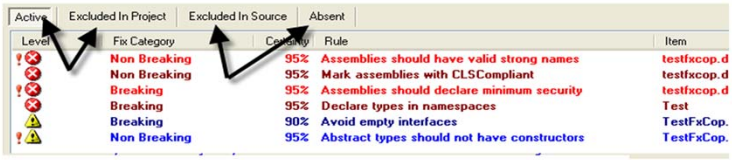
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 14

2.10: FxCop Analysis Result

FxCop Analysis Result

- Active – Current issues
- Excluded In Project – Issue suppressed for the entire project
- Excluded In Source – Individual issue suppressed at the source level
- Absent – Issues that have been resolved



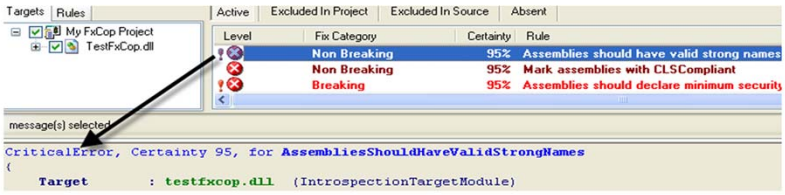
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 15

2.10: FxCop Analysis Result

FxCop Analysis Result (contd..)

- Levels
  - Critical Error
  - Error
  - Critical Warning
  - Warning
  - Informational



2.10: FxCop Analysis Result

FxCop Analysis Result (contd..)

- Fix Category
- Breaking – resolving this issue will cause dependent code to break; recompile needed
  - Non Breaking – issue can be resolved without breaking dependent code
- Certainty – How sure it is an issue

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 17

2.10: FxCop Analysis Result

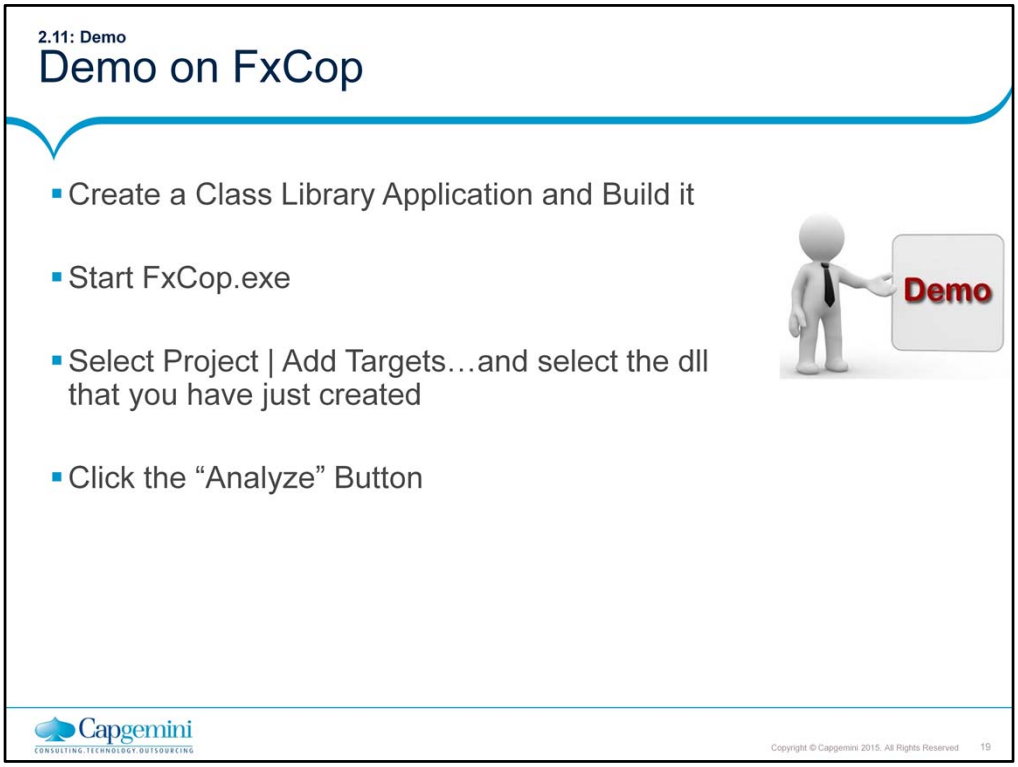
FxCop Analysis Result (contd..)

- Details pane for a selected issue
  - Where the issue is located, brief description, link to more info, etc

```
CriticalError, Certainty 95, for AssembliesShouldHaveValidStrongNames
{
    Target       : testfxcop.dll  (IntrospectionTargetModule)
    Resolution   : "Sign 'TestFxCop' with a strong name key."
    Help         : http://www.gotdotnet.com/team/fxcop/docs/rules.aspx?version=1.35&url=/Design/.
    Category     : Microsoft.Design  (String)
    CheckId      : CA2210  (String)
    RuleFile     : Design Rules  (String)
    Info         : "Either the assembly has no strong name, an invalid
                   one, or the strong name is valid only because of the
                   computer configuration. The assembly should not be
                   deployed in this state. The most common causes of this
                   are: 1) The assembly's contents were modified after
                   it was signed. 2) The signing process failed. 3) The
                   assembly was delay-signed. 4) A registry key existed
                   that allowed the check to pass (where it would not
                   have otherwise)."
    Created      : 10/15/2010 5:08:54 AM  (DateTime)
    LastSeen     : 10/15/2010 5:08:54 AM  (DateTime)
    Status       : Active  (MessageStatus)
    Fix Category : NonBreaking  (FixCategories)
}
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 18



Demonstration: HelloWorld:
Let us take a look at the first ASP.NET application. To create this application follow the steps given below:

Step 1: Open Visual Studio 2013. Select **File → New-WebSite**. It opens the **New WebSite** dialog box.
With ASP.NET 4.5, using Visual Studio 2013 you have an option to create an application with virtual directory mapped to IIS or a standalone application outside the confines of IIS.
Built-in Web Server: As mentioned earlier ASP.Net 4.5 comes with a built-in Web Server called as **Cassini**. By default, VS2013 builds applications without the use of IIS. The location provided by default for your application is 'C:\Documents and Settings\username\My Documents\Visual Studio 2013\WebSites'. Optionally you can also create your own folder and allow your WebSites to be created in that location. If you use the Built-in Web Server, then you are not locked into the Websites folder or in the 'C:\Inetpub\wwwroot' folder. You can also run your application completely from this location. Hence with this new way of creating ASP.NET applications you are not dependent on having access to any IIS server and you can go ahead and develop applications on any Windows OS machine.
The figure on the following page shows the creation of website on the FileSystem, that is in the Built-in Web Server.

2.12: What is StyleCop?

What is StyleCop?

- StyleCop is a source analysis tool for C#
- It can be used for analyzing source code as opposed to compiled assemblies which is the area for FxCop
- StyleCop is currently in version 4.3 and can be downloaded from http://stylecop.codeplex.com
- It has been used by Microsoft to help teams enforce a common set of best practices for layout, readability, maintainability, and documentation of C# source code

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 20

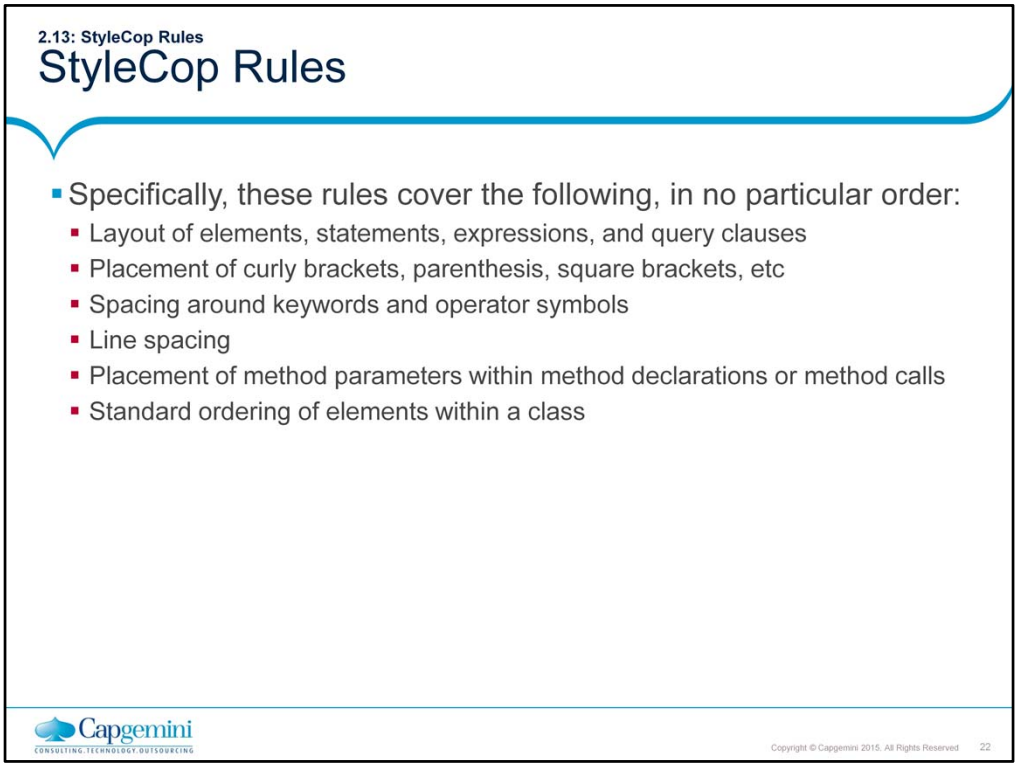
2.12: What is StyleCop?

What is StyleCop? (contd..)

- StyleCop is similar in many ways to Microsoft Code Analysis (specifically FxCop), but there are some important distinctions
- FxCop performs its analysis on compiled binaries, while StyleCop analyzes the source code directly
- For this reason, FxCop focuses more on the design of the code, while StyleCop focuses on layout, readability and documentation

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 21


2.13: StyleCop Rules
StyleCop Rules
Specifically, these rules cover the following, in no particular order:
Layout of elements, statements, expressions, and query clauses
Placement of curly brackets, parenthesis, square brackets, etc
Spacing around keywords and operator symbols
Line spacing
Placement of method parameters within method declarations or method calls
Standard ordering of elements within a class
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved
22

2.13: StyleCop Rules

StyleCop Rules (contd..)

- Formatting of documentation within element headers and file headers
- Naming of elements, fields and variables
- Use of the built-in types
- Use of access modifiers
- Allowed contents of files
- Debugging text

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

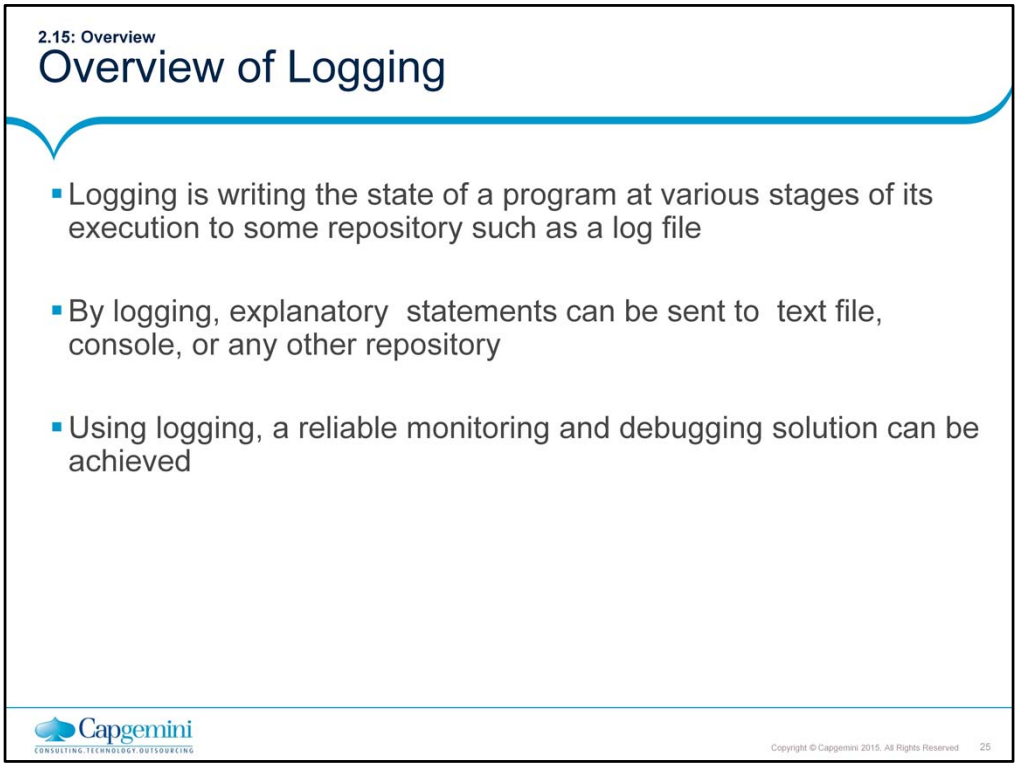
 23

2.14: Demo

Demo on StyleCop

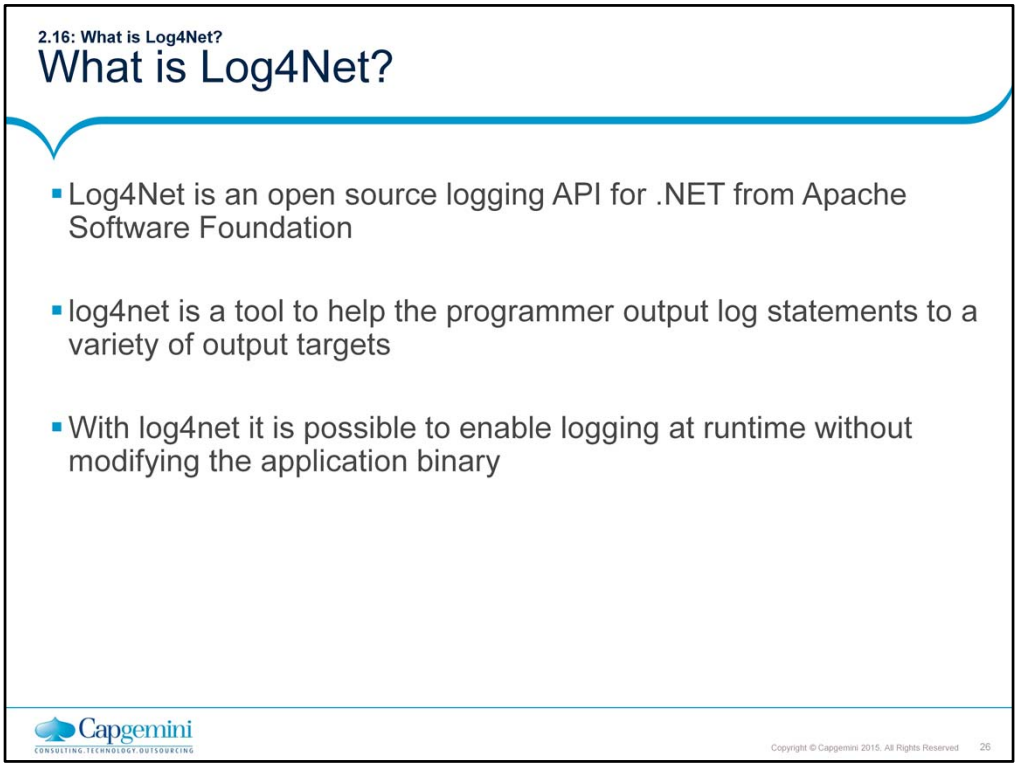
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 24

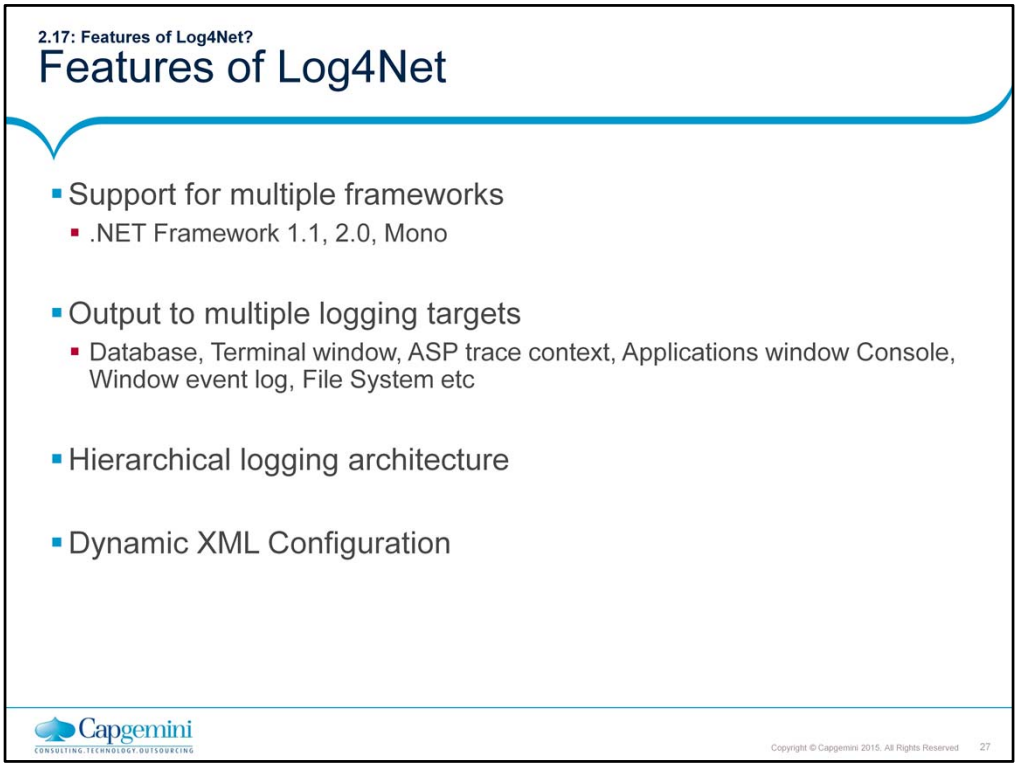


Logging is used due to the following reasons:

It can be used for debugging.
It is cost effective than including some debug flag.
There is no need to recompile the program to enable debugging.
It does not leave your code messy.
Priority levels can be set.
Log statements can be appended to various destinations such as file, console, socket, database, and so on.
Logs are needed to quickly target an issue that occurred in service.

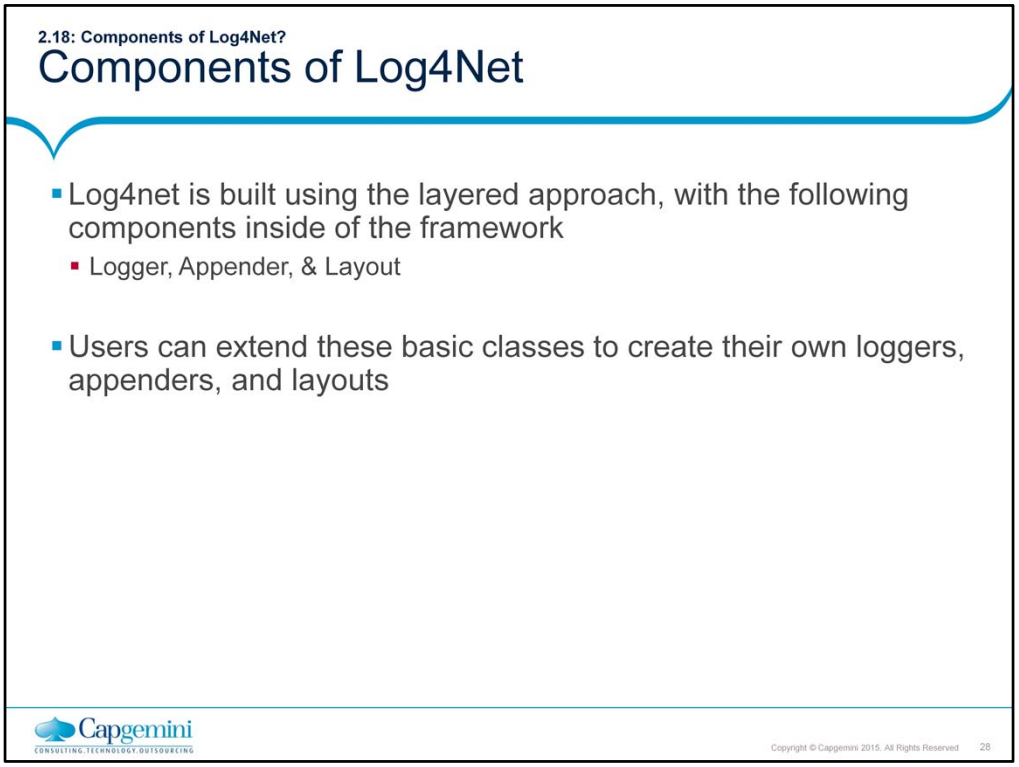


log4net provides many advantages over other logging systems, which makes it a perfect choice for use in any type of application, from a simple single-user application to a complex multiple-threaded distributed application using remoting



Hierarchical logging is an ideal fit with component based development. Each component has its own of logger. When individually tested, the properties of these loggers may be set as the developer requires. When combined with other components, the loggers inherit the properties determined by the integrator of the components. One can selectively elevate logging priorities on one component without affecting the other components. This is useful when you need a detailed trace from just a single component without crowding the trace file with messages from other components. All this can be done through configuration files; no code changes are required.

log4net is configured using an XML configuration file. The configuration information can be embedded within other XML configuration files (such as the application's .config file) or in a separate file. The configuration is easily readable and updateable while retaining the flexibility to express all configurations. Alternatively log4net can be configured programmatically.



The Logger is the main component with which your application interacts. It is also the component that generates the log messages.
Generating a log message is different than actually showing the final output. The output is showed by the Layout component.
The logger provides you with different methods to log any message. You can create multiple loggers inside of your application.
Any good logging framework should be able to generate output for multiple destinations, such as outputting the trace statements to the console or serializing it into a log file. log4net is a perfect match for this requirement. It uses a component called *Appender* to define this output medium. As the name suggests, these components append themselves to the Logger component and relay the output to an output stream. You can append multiple appenders to a single logger.
The Layout component is used to display the final formatted output to the user. The output can be shown in multiple formats, depending upon the layout we are using. It can be linear or an XML file. The layout component works with an appender. There is a list of different layouts in the API documentation. You cannot use multiple layouts with an appender. To create your own layout, you need to inherit the log4net.Layout.LayoutSkeleton class, which implements the ILayout interface.

2.19: Demo

Demo on Log4Net

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 29

Summary

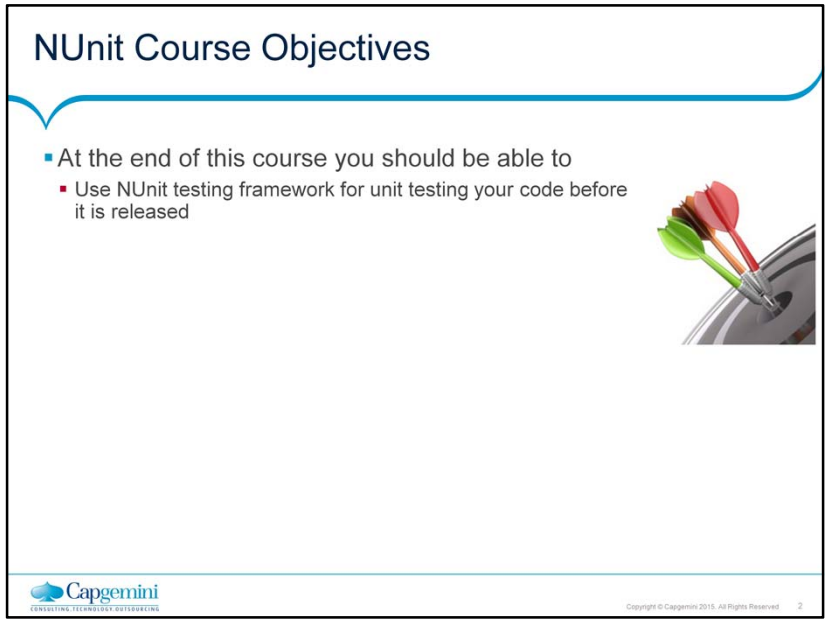
- In this lesson, you have learnt:
  - What is the use of FxCop as a Code Analysis Tool
  - How to use the FxCop tool to detects errors in Managed Code
  - StyleCop as a Source Code Analysis Tool
  - How to use StyleCop

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 30

Developer Tools for .NET

Lesson 1: Unit Testing with NUnit & Overview of NCover

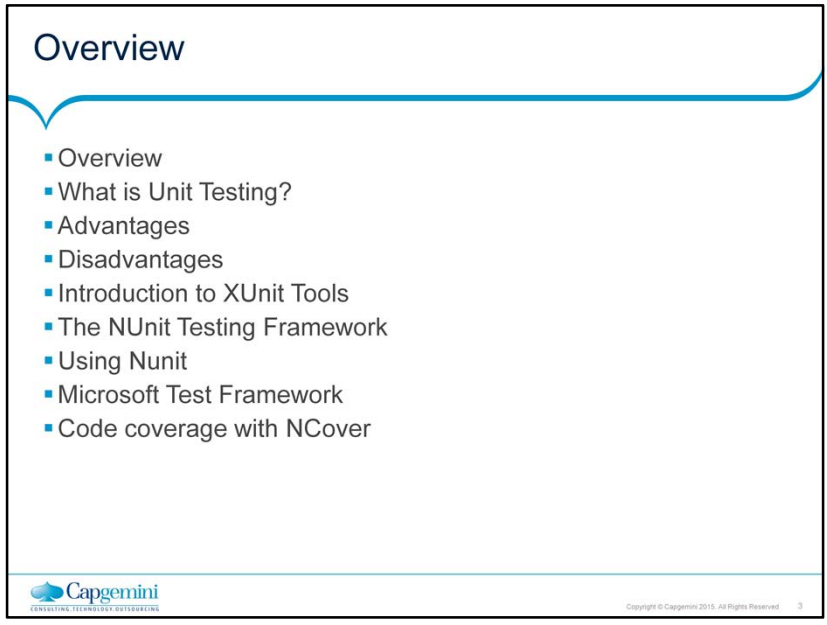


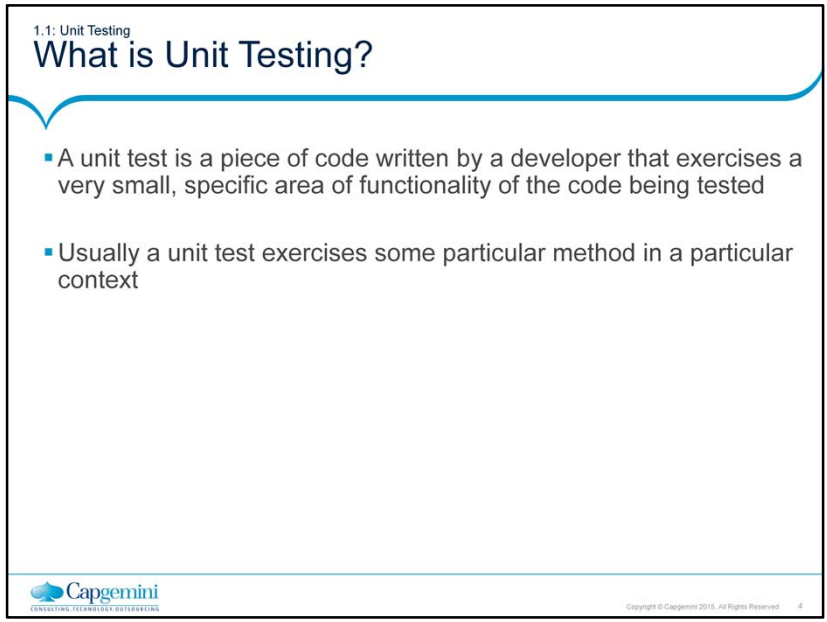
Goals one is going to attain at the end of course is being able to use NUnit testing framework for unit testing your code before it is released.
What is Unit Testing?
Testing of individual software components. Each module is tested alone in an attempt to discover any errors in its code. In computer programming, a unit test is a method of testing the correctness of a particular module of source code.

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. Unit testing has proven its value in that a large percentage of defects are identified during its use.

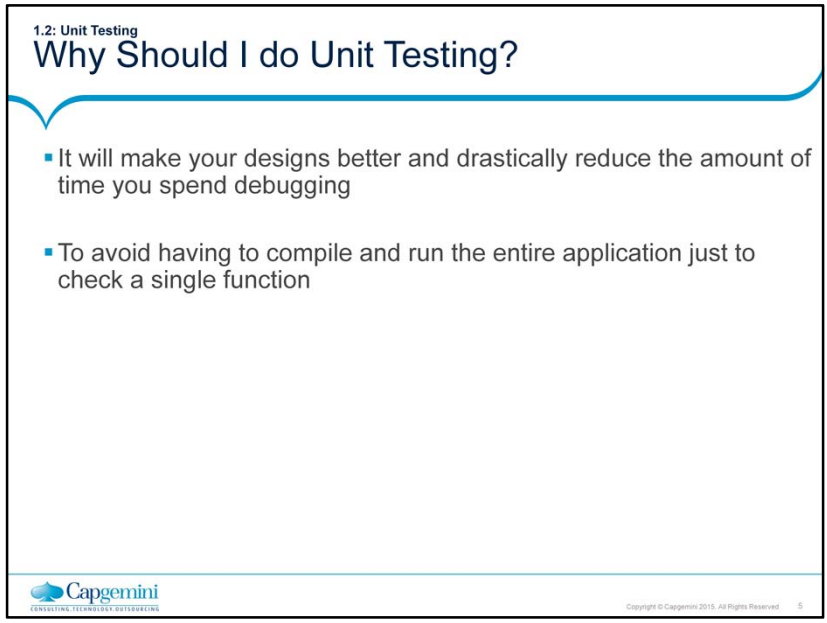
Qualities of Good Unit Test:
These run fast.
These help to localize problems.

Why do we need Unit Testing?
Unit tests find problems early in the development cycle.
Developers will be less afraid to change existing code.
The development process becomes more flexible.
Software development will become more predictable and repeatable

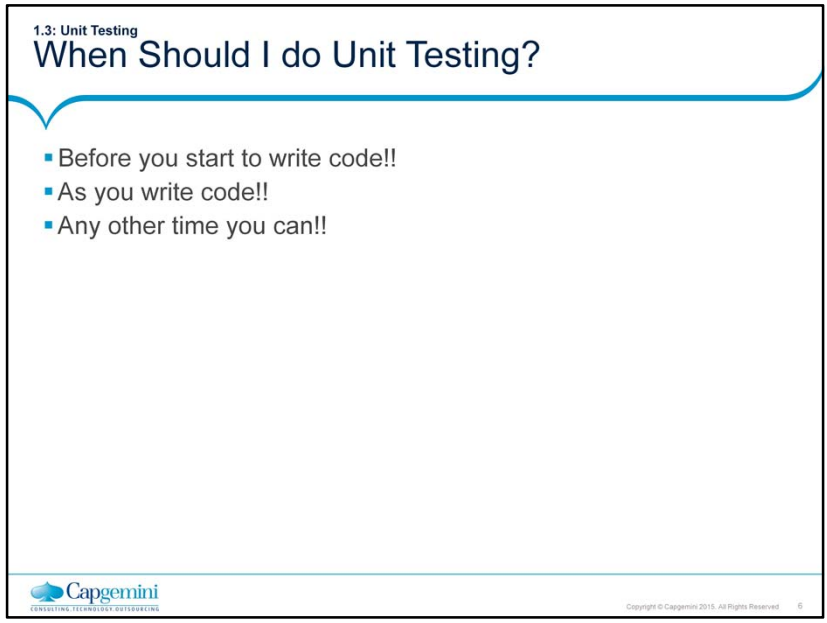

Overview
Overview
What is Unit Testing?
Advantages
Disadvantages
Introduction to XUnit Tools
The NUnit Testing Framework
Using Nunit
Microsoft Test Framework
Code coverage with NCover
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved 3

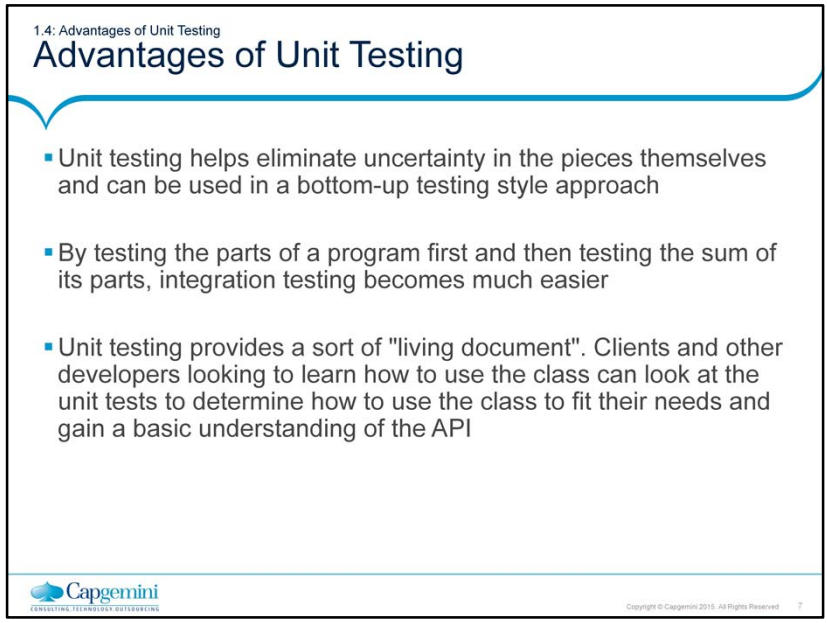


Unit Tests are "programs written to run in batches and test classes. Each typically sends a class a fixed message and verifies it returns the predicted answer." In practical terms this means that you write programs that test the public interfaces of all of the classes in your application. This is not requirements testing or acceptance testing. Rather it is testing to ensure the methods you write are doing what you expect them to do. This can be very challenging to do well. First of all, you have to decide what tools you will use to build your tests. In the past we had large testing engines with complicated scripting languages that were great for dedicated QA teams, but weren't very good for unit testing. What journeyman programmers need is a toolkit that lets them develop tests using the same language and IDE that they are using to develop the application. Most modern Unit Testing frameworks are derived from the framework created by Kent Beck for the first XP project, the Chrysler C3 Project. It was written in Smalltalk and still exists today, although it has gone through many revisions. Later, Kent and Erich Gamma (of Patterns fame) ported it to Java and called it jUnit. Since then, it has been ported to many different languages, including C++, VB, Python, Perl and more.

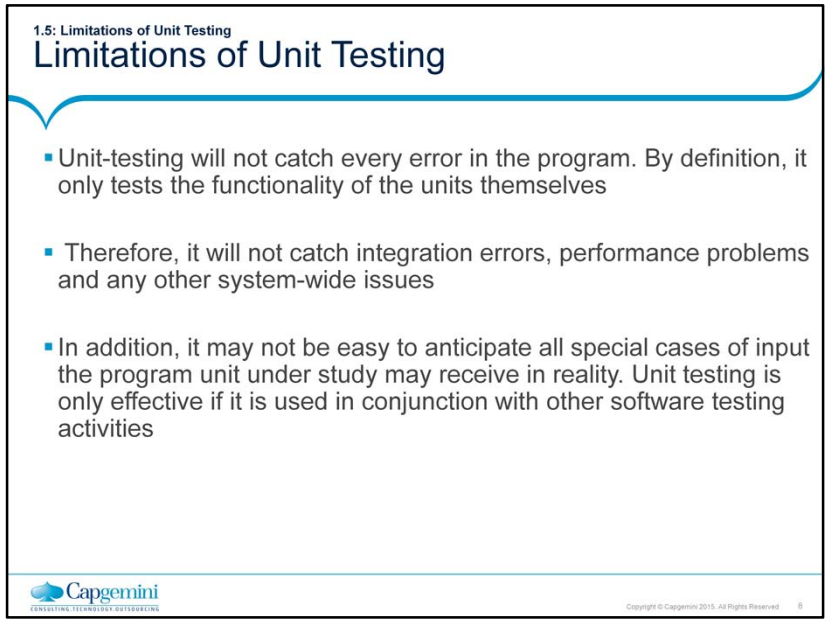


This reduces the cycle time between writing code and testing it. The shorter this cycle the better able you are to test specific functionality.


1.3: Unit Testing
When Should I do Unit Testing?
Before you start to write code!!
As you write code!!
Any other time you can!!
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved
6



Because some classes may have references to other classes, testing a class can frequently spill over into testing another class. A common example of this is classes that depend on a database: in order to test the class, the tester often writes code that interacts with the database. This is a mistake, because a unit test should never go outside of its own class boundary. As a result, the software developer abstracts an interface around the database connection, and then implements that interface with their own mock object. This results in loosely coupled code, minimizing dependencies in the system



It is unrealistic to test all possible input combinations for any non-trivial piece of software. A unit test can only show the presence of errors; it cannot show the absence of errors.

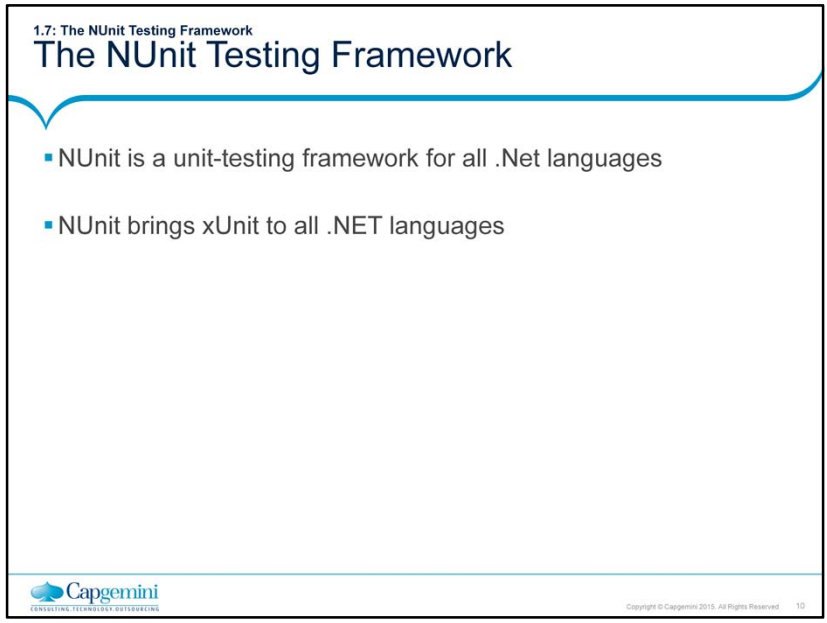
1.6: XUnits Tools

XUnits Tools

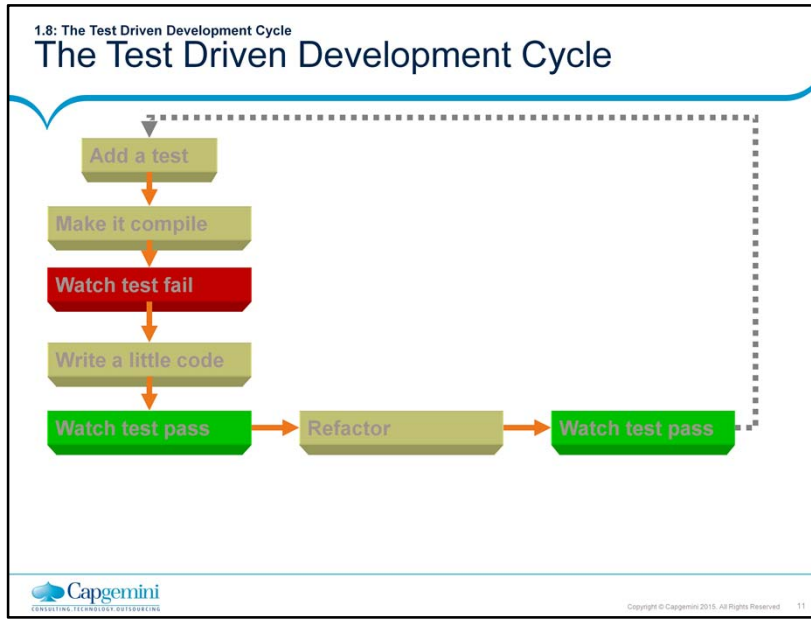
- In order to run tests in the code, people typically use XUnit tools
- JUnit for Java, NUnit for .NET (nunit.org)
- These tools often come in two flavors: command-line and GUI
- XUnit tools allow you to run all your tests and see which passed and which failed
- Best of all, these tools are free

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 9

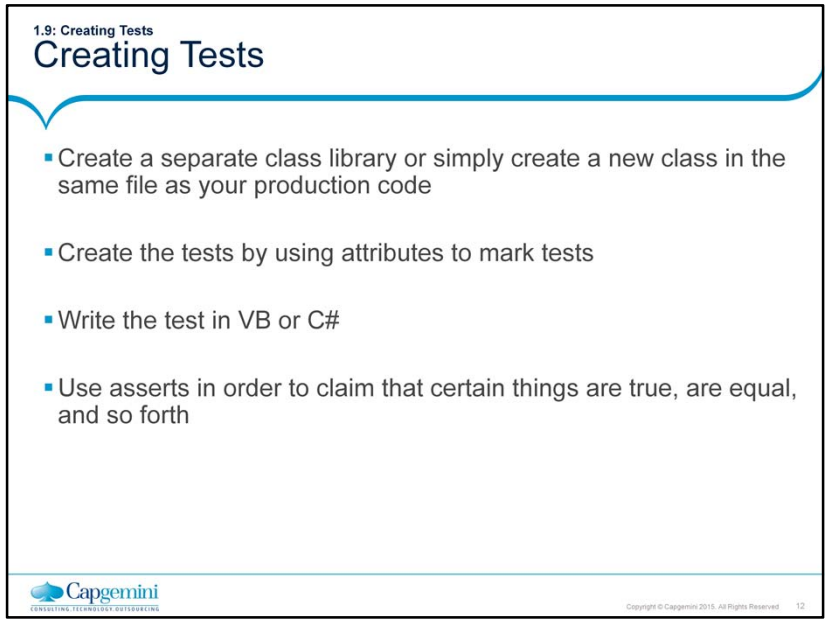


NUnit is an application designed to facilitate unit testing. It consists of both a command line and Window's interface, allowing it to be used both interactively and in automated test batches or integrated with the build process.

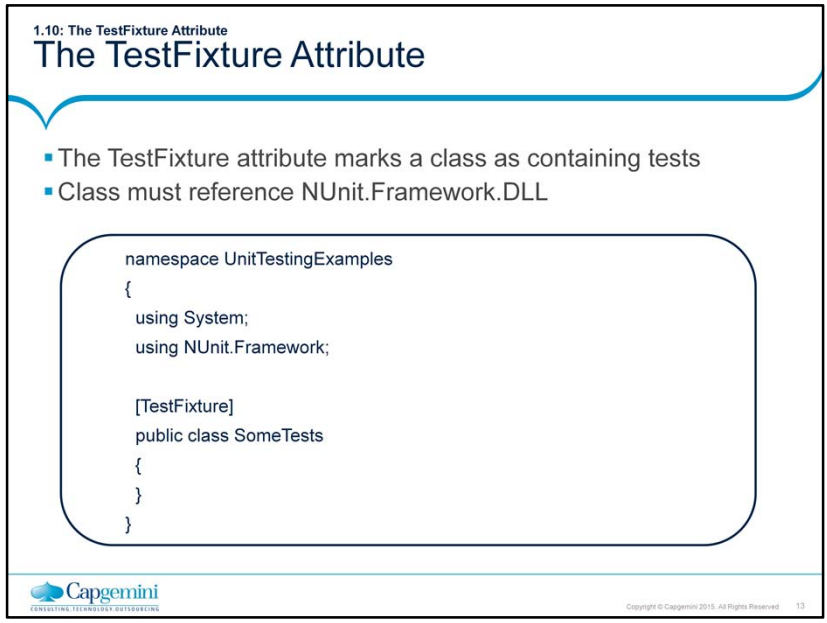


Instead of writing functional code first and then your testing code as an afterthought, if you write it at all, you instead write your test code before your functional code. Furthermore, you do so in very small steps – one test and a small bit of corresponding functional code at a time. A programmer taking a TDD approach refuses to write a new function until there is first a test that fails because that function isn't present.

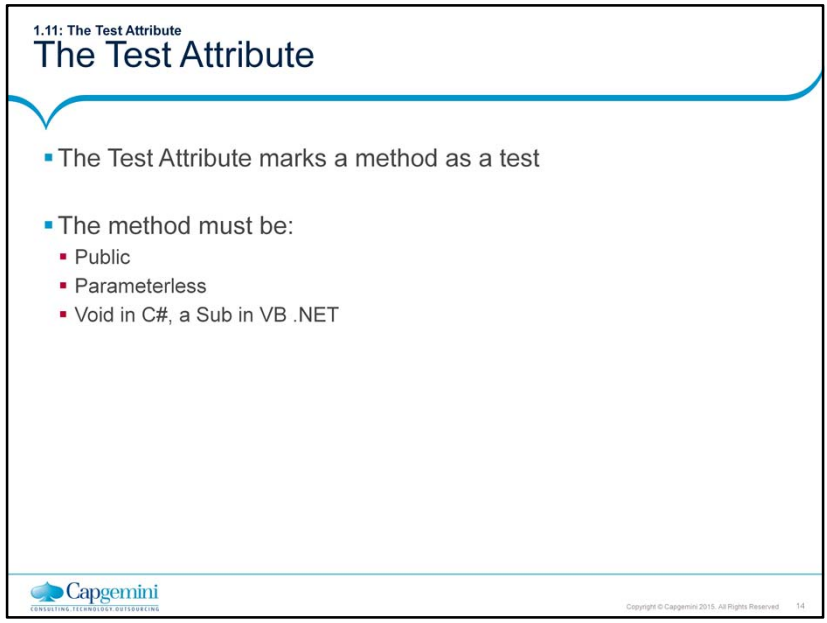
Why TDD?
A significant advantage of TDD is that it enables you to take small steps when writing software.



Assertions are central to unit testing in any of the xUnit frameworks, and NUnit is no exception. NUnit provides a rich set of assertions as static methods of the Assert class.
If an assertion fails, the method call does not return and an error is reported. If a test contains multiple assertions, any that follow the one that failed will not be executed. For this reason, it's usually best to try for one assertion per test.



The TestFixture attribute designates that a class is a test fixture. Classes thus designated contain setup, teardown, and unit tests.



The Test attribute indicates that a method in the test fixture is a unit test. The unit test engine invokes all the methods indicated with this attribute once per test fixture, invoking the set up method prior to the test method and the tear down method after the test method, if they have been defined.

The test method signature must be specific: public void xxx(), where "xxx" is a descriptive name of the test. In other words, a public method taking no parameters and returning no parameters.

Upon return from the method being tested, the unit test typically performs an assertion to ensure that the method worked correctly.

1.11: The Test Attribute

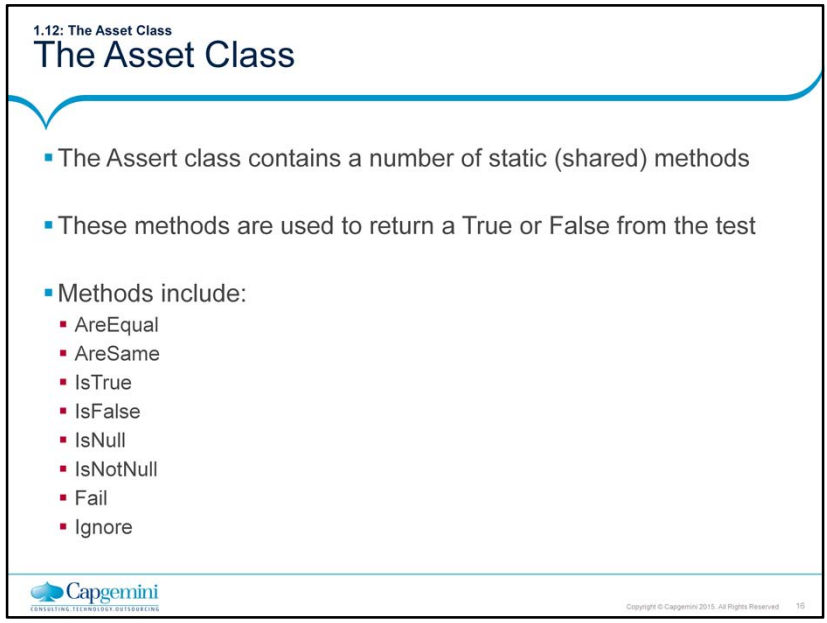
The Test Attribute (contd..)

- The following code illustrates the use of this attribute

```
[TestFixture]
  public class SomeTests
  {
    [Test]
    public void TestOne()
    {
      // Do something...
    }
  }
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 15


1.12: The Asset Class
The Asset Class
The Assert class contains a number of static (shared) methods
These methods are used to return a True or False from the test
Methods include:
AreEqual
AreSame
IsTrue
IsFalse
IsNull
IsNotNull
Fail
Ignore
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved
16

1.13: Example of a Simple Test

Example: A Simple Test

```
[TestFixture]
public class MyTests
{
  [Test]
  public void AddTest()
  {
    int sum=myMath.Add(2,3);
    Assert.AreEqual(5,sum);
  }
}
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 17

1.14: Running Tests

Running Tests

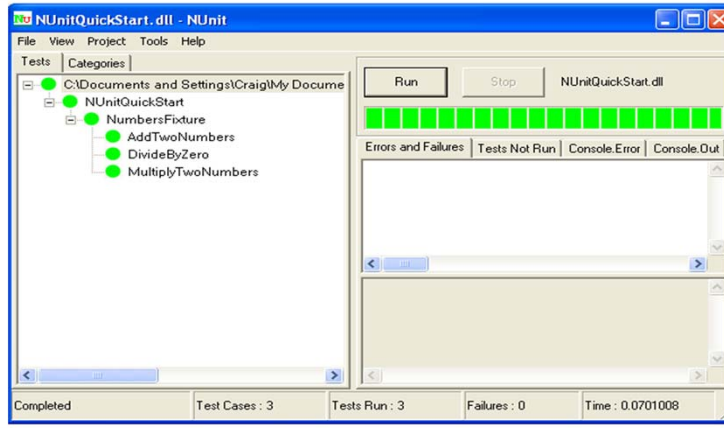
- NUnit comes with two different Test Runner applications: a Windows GUI app and a console XML app
- To use the GUI app, just run the application and tell it where your test assembly resides
- The test assembly is the class library (or executable) that contains the Test Fixtures
- The app will then show you a graphical view of each class and test that is in that assembly
- To run the entire suite of tests, simple click the Run button

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 18

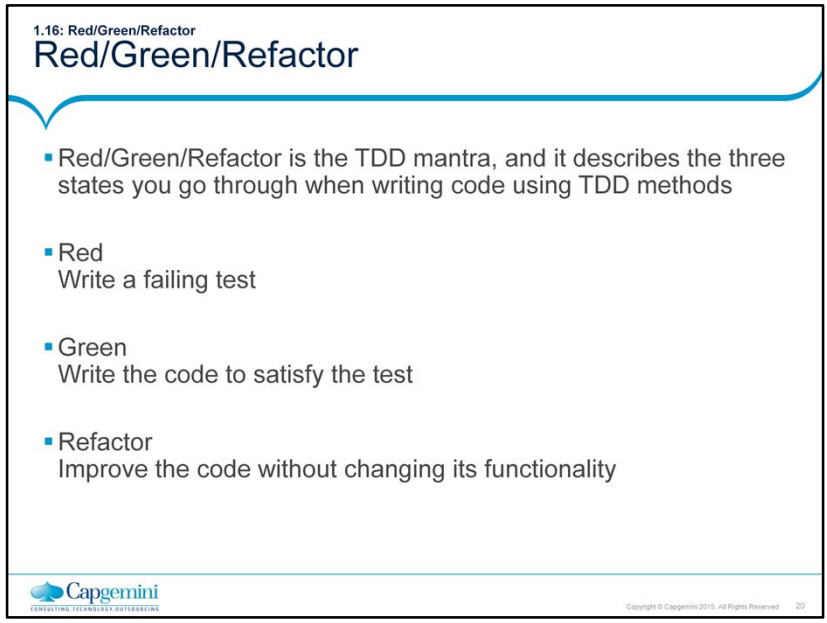
1.15: NUnit GUI Screen

NUnit GUI Screen



Capgemini

 19


1.16: Red/Green/Refactor
Red/Green/Refactor
Red/Green/Refactor is the TDD mantra, and it describes the three states you go through when writing code using TDD methods
Red
Write a failing test
Green
Write the code to satisfy the test
Refactor
Improve the code without changing its functionality
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved 20

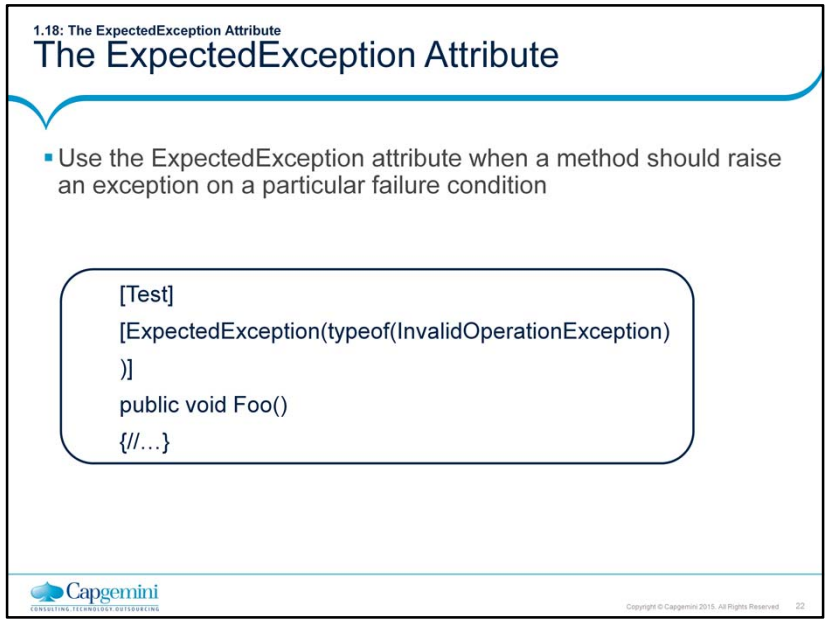
1.17: Demo

Demo on Using NUnit Framework

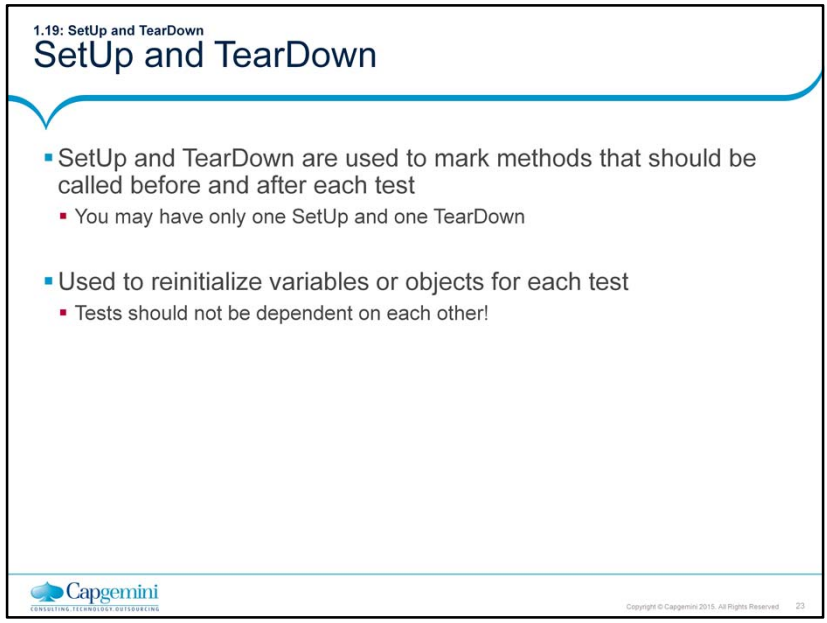
- Using NUnit Framework

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 21

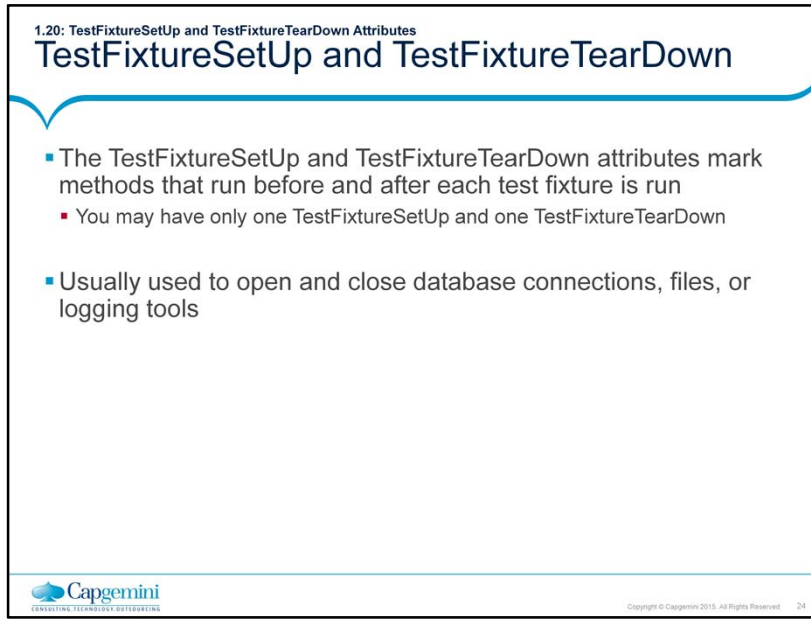


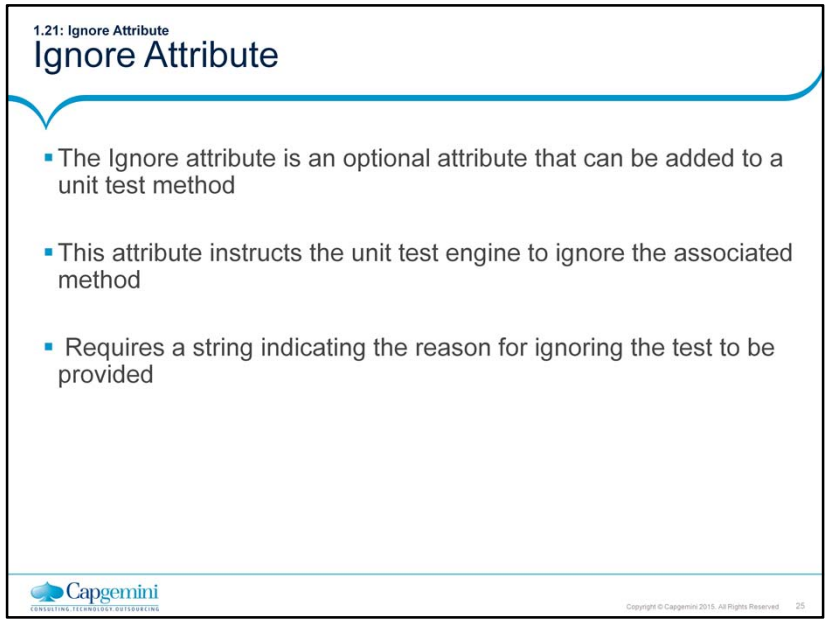
The ExpectedException attribute is an optional attribute that can be added to a unit test method (designated using the Test attribute).
As unit testing should in part verify that the method under test throws the appropriate exceptions, this attribute causes the unit test engine to catch the exception and pass the test if the correct exception is thrown.



The SetUp attribute is associated with a specific method inside the test fixture class. It instructs the unit test engine that this method should be called prior to invoking each unit test. A test fixture can only have one SetUp method.

The TearDown attribute is associated with a specific method inside the test fixture class. It instructs the unit test engine that this method should be called after invoking each unit test. A test fixture can only have one TearDown method.


1.20: TestFixtureSetUp and TestFixtureTearDown Attributes
TestFixtureSetUp and TestFixtureTearDown
The TestFixtureSetUp and TestFixtureTearDown attributes mark methods that run before and after each test fixture is run
You may have only one TestFixtureSetUp and one TestFixtureTearDown
Usually used to open and close database connections, files, or logging tools
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved
24



You probably won't use this attribute very often, but when you need it, you'll be glad it's there. If you need to indicate that a test should not be run, use the Ignore attribute.
If you feel the need to temporarily comment out a test, use this instead. It lets you keep the test in your arsenal and it will continually remind you in the test runner output.

1.21: Ignore Attribute

Ignore Attribute (contd..)

```
[TestFixture]
  public class SomeTests
  {
    [Test]
    [Ignore("We're skipping this one for now.")]
    public void TestOne()
    {
      // Do something...
    }
  }
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 26

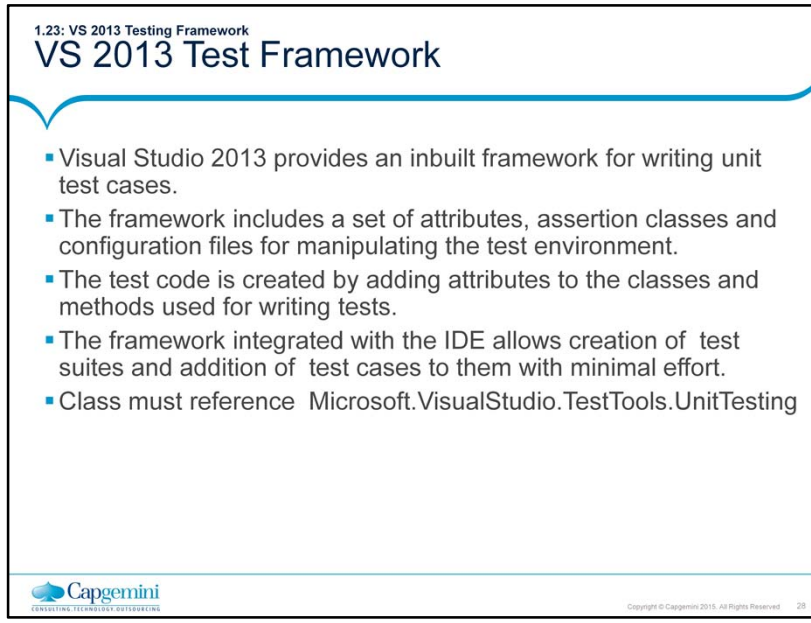
1.22: Demo

Demo

- Extending the Simple Example of TDD and NUnit
  - Adding setup and teardown routines
  - Handling expected exceptions

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 27



The Microsoft.VisualStudio.TestTools.UnitTesting namespace exposes a host of predefined attributes and classes. For example, the Assert class contains methods which can be used to verify conditions in unit testing.

An Assertion is a Boolean expression that describes what must be true when some action has been executed.

If an assertion method fails, the method call does not return, and an error is reported. Hence, if a test contains multiple assertions, any assertions following the one that has failed will not be executed. It's usually best to try for one assertion per test.

A test case is the smallest unit required to test a functionality. We need to identify initial conditions, test actions and expected output for each test case.

A test suite is a group of related test cases.

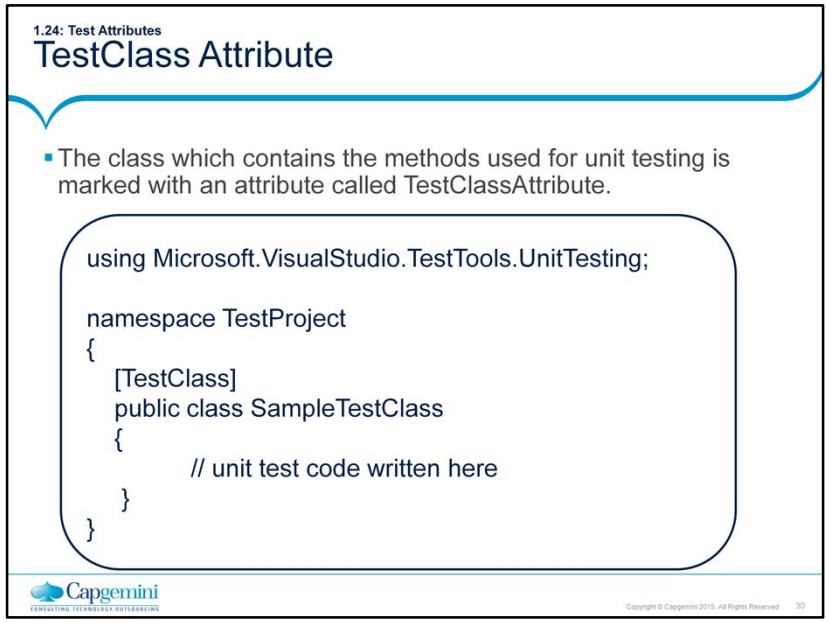
1.23: VS 2013 Testing Framework

Creating Tests Using Visual Studio 2013

- Create a separate class library or simply create a new class in the same project which contains the code to be tested.
- Create the tests by using predefined attributes.
- Write the test case code .
- Use appropriate Asserts.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 29



Add a reference to Microsoft.VisualStudio.TestTools.UnitTesting namespace to use the above mentioned attribute.
The class which contains the above attribute is called a Test Class , and will contain the unit test code.
Test Classes must be public, with a public parameter less constructor.

1.24: Test Attributes

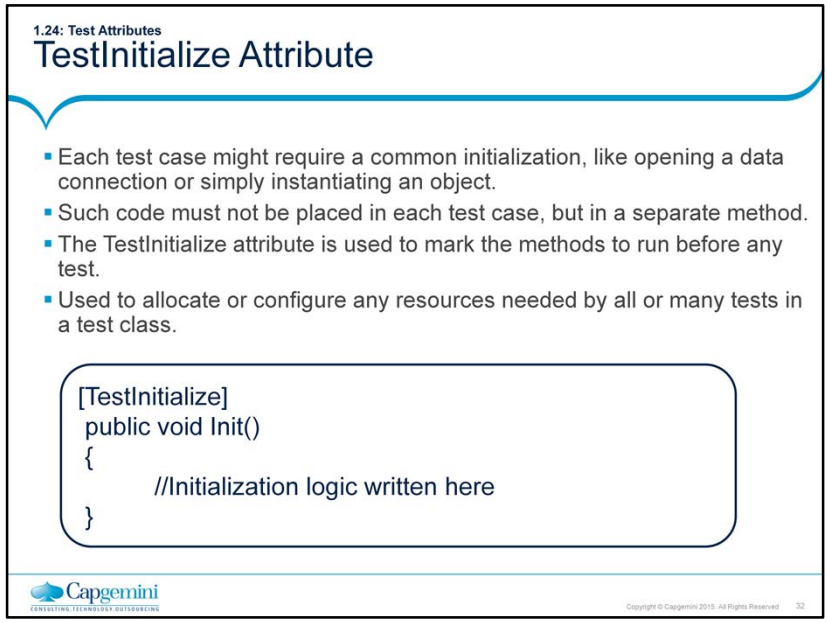
TestMethod Attribute

- The TestMethodAttribute is used to mark a method as a test method.
- A method marked with this attribute must be public, void and without parameters.

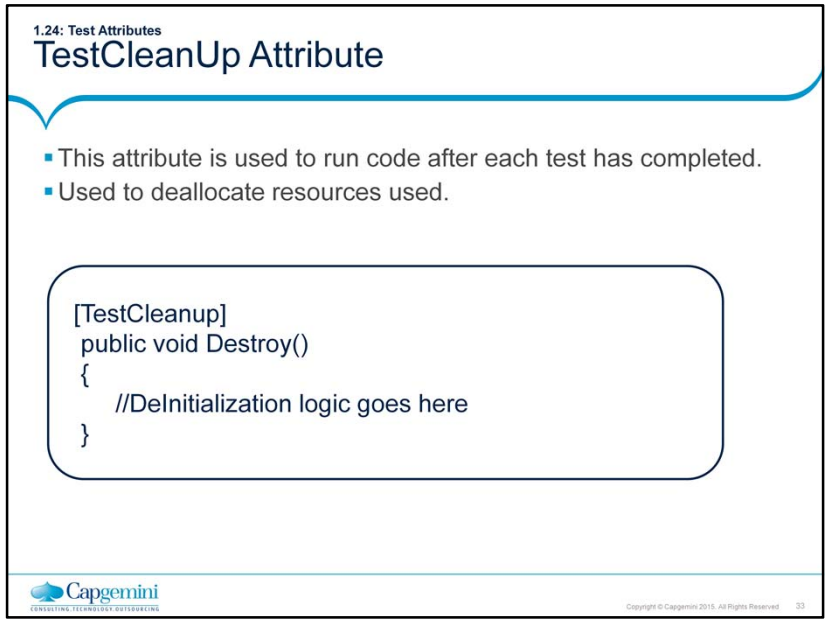
```
[TestClass]
public class SampleTestClass
{
    [TestMethod]
    public void TestOne()
    {
        //Unit Test logic written here
    }
}
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

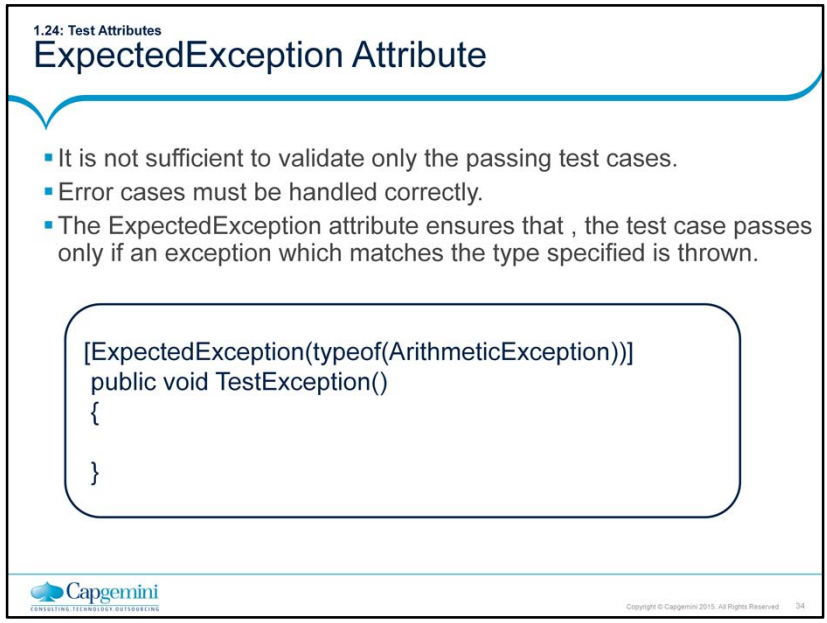
 31



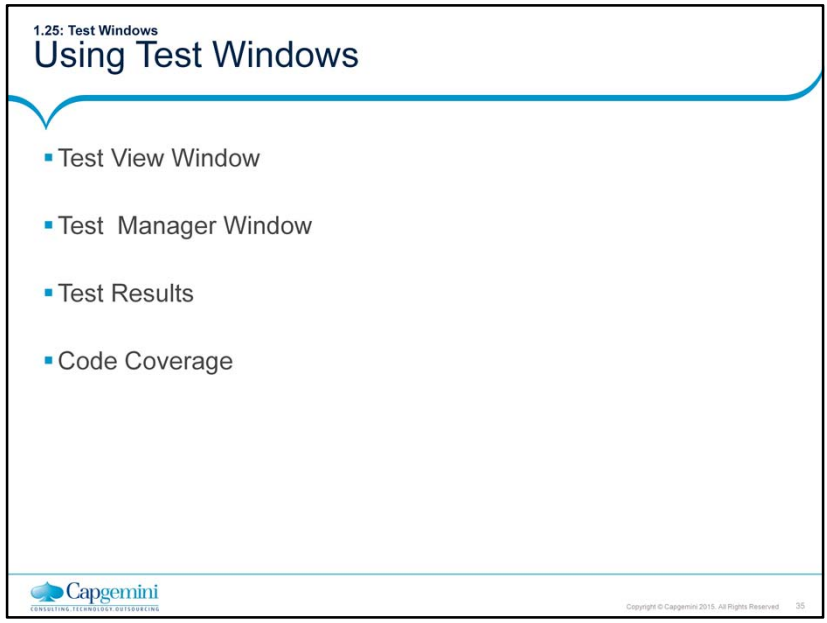
Any method marked with this attribute will run once for every iteration in the test. If you need to do initialization operations once, that apply to the entire test, use the ClassInitializeAttribute.



Use TestCleanup to run code after each test has run. If you need to perform clean up operations once, that apply to the entire test, use the ClassCleanupAttribute.



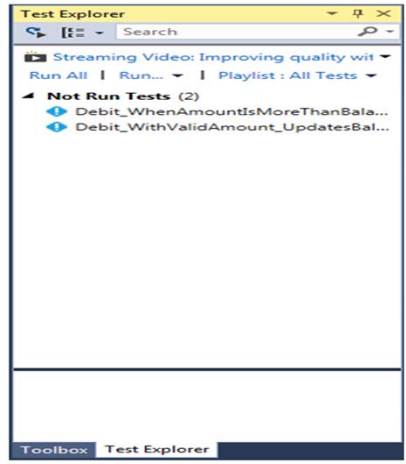
You can also overload the attribute constructor with a string, to specify the error message which must be reported.



a. The Test View Window: this window shows a list of all loaded tests. You can use this window for running , grouping or filtering tests. The window can be opened through the Test -> Windows Menu Item (at the top of the IDE).
b. The Test Manager Window: Similar to Test View, but can organize tests. Used to show the list of tests, the list of loaded tests, and the list of unorganized tests.
c. The Test Results Window: used to view test results. You can also re run a test, pause/ stop a test. The window exposes functionality to publish or import test reports using TFS (if connected).
d. The Code Coverage Window: shows the percentage of code covered by the tests. Uses coloring to indicate the code covered, partially covered or not covered by the test case.

1.25: Test Windows

Using Test Windows



Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 36

1.26: Demo

Demo

- Creating a test project in VS 2013

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 37

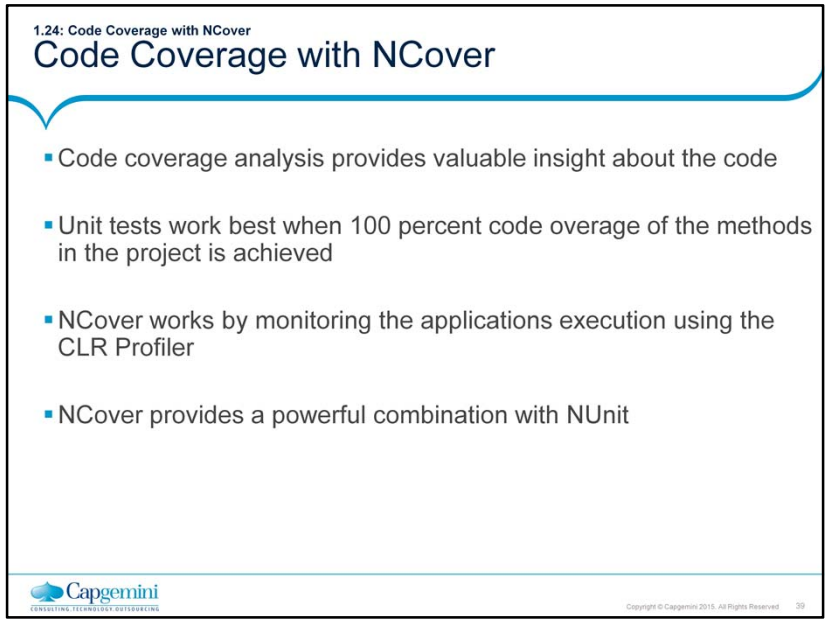
1.23: Programming Sequence

The Typical Programming Sequence

- Write a test
- Run the test. It fails to compile because the code you're trying to test doesn't even exist yet!
- Write a bare-bones stub to make the test compile
- Run the test. It should fail. (If it doesn't, then the test wasn't very good.)
- Implement the code to make the test pass
- Run the test. It should pass. (If it doesn't, back up one step and try again.)
- Start over with a new test!

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 38


1.24: Code Coverage with NCover
Code Coverage with NCover
Code coverage analysis provides valuable insight about the code
Unit tests work best when 100 percent code overage of the methods in the project is achieved
NCover works by monitoring the applications execution using the CLR Profiler
NCover provides a powerful combination with NUnit
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved
39

1.25: Demo on Using NCover

Demo on Using NCover

- Using NCover

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 40

Summary

- NUnit provides a powerful framework for unit testing
- Developers assisted in implementing repeatable, best-practice processes

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Developer Tools for .NET

Lab Guide

Document Revision History

Date	Revision No.	Author	Summary of Changes
15-May-2016	1	Nachiket Inamdar	Added Microsoft Test Framework content.

Table of Contents

Lab 1: Using NUnit to Test Code

Goals	• Understand working of NUnit to Test the code • Understand the steps involved in writing correct code.
Time	90 Minutes
Lab Setup	VS.NET 2013 & NUnit 2.x Framework

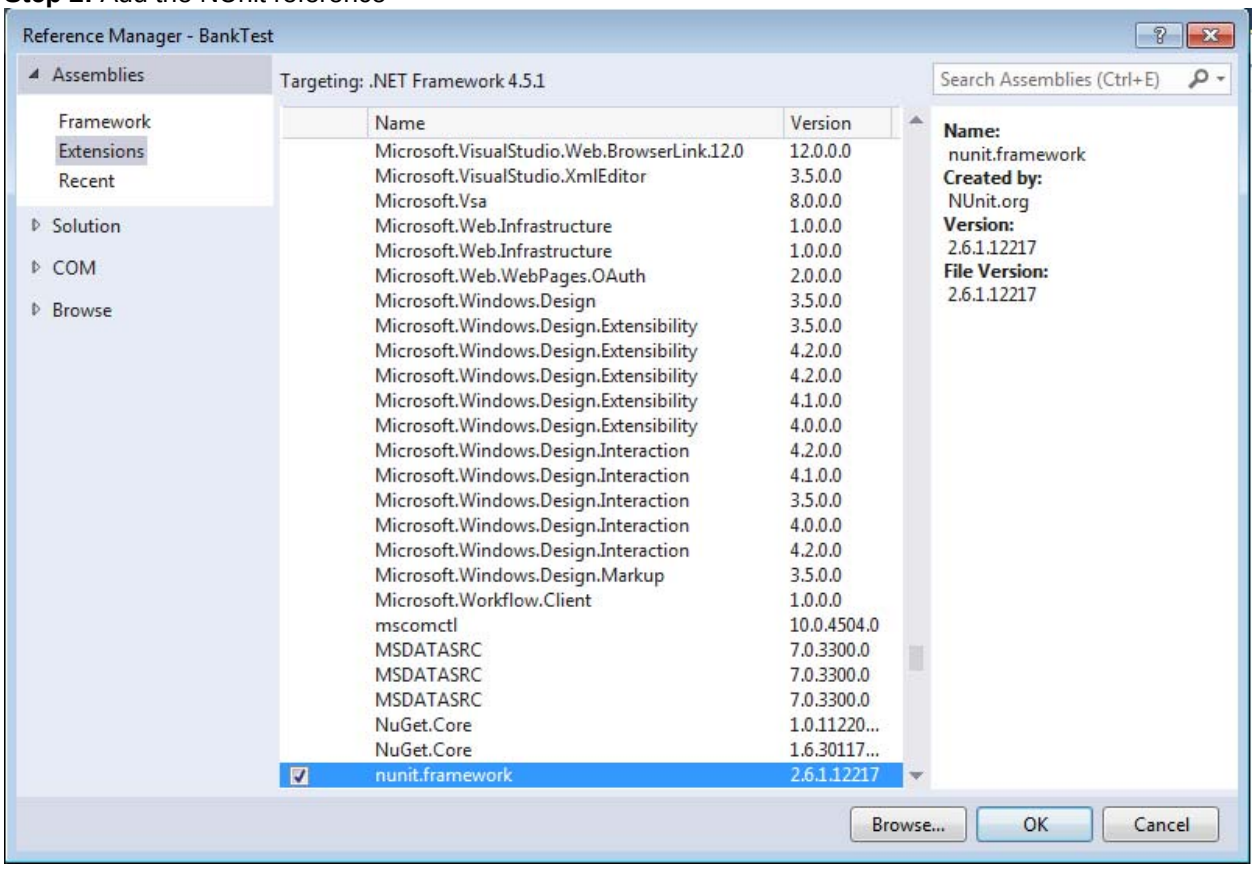
Lab 1 A. Problem statement

Write the production code & check the same for correctness.

Solution:

Step 1: Create your project of Library Type.

Step 2: Add the NUnit reference



Step 3: Add your TestCase

1. Add the reference in your code to the NUnit namespace:

<< to do>>

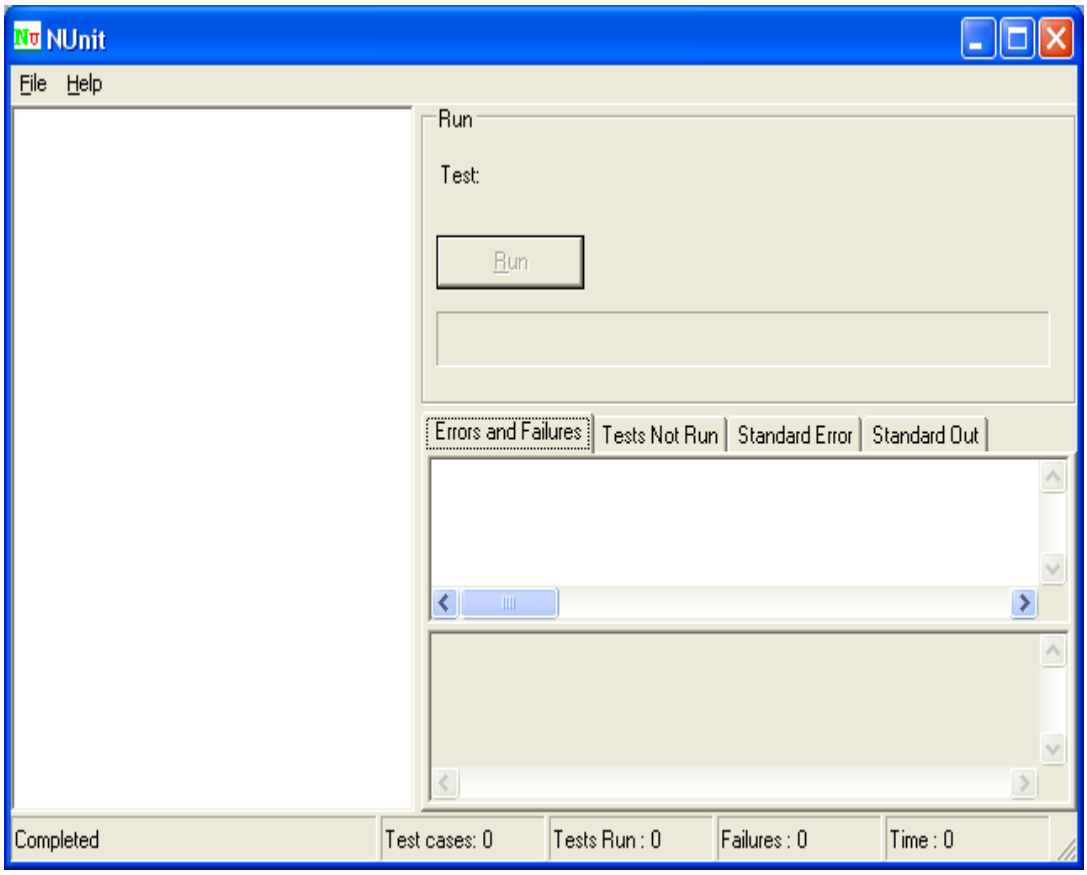
2. Add your testing class to your source file:

```
namespace TestDemo1
{
        [TestFixture]
        public class MyTestClass
        {
                [Test]
                public void TestFunction()
                {

                }
        }
}
```

3. Compile your project.
<< to do >>

Step 4: Starting NUnit-GUI

1. Start the NUnit-GUI application
<<to do>>



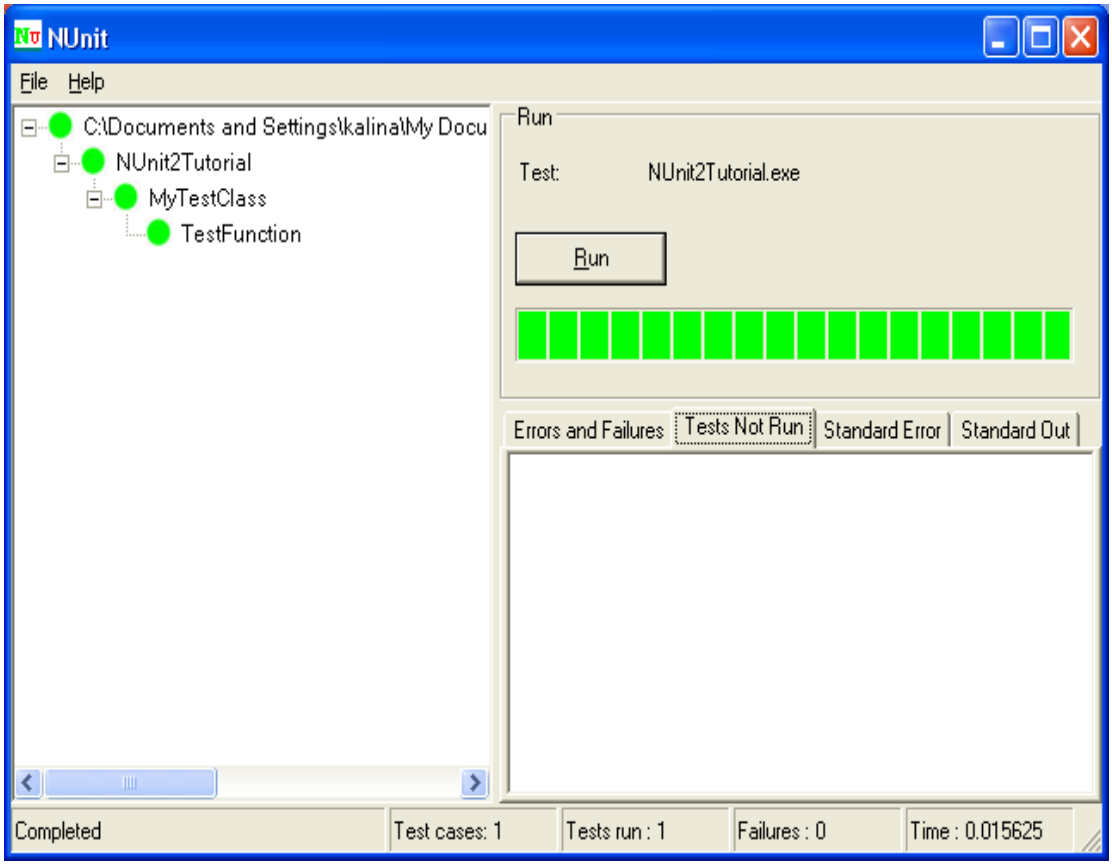
2. Select the library of your project in order to run the containing tests.

<< to do>>

You will immediately notice that NUnit2 is looking through the complete compilation unit and finds any TestFixtures. In our case the TestFixture is called MyTestClass and the only test is TestFunction.

Step 5: Testing with NUnit-GUI

Click on the “Run” button.



You will see a green bar which tells you that all tests succeeded.

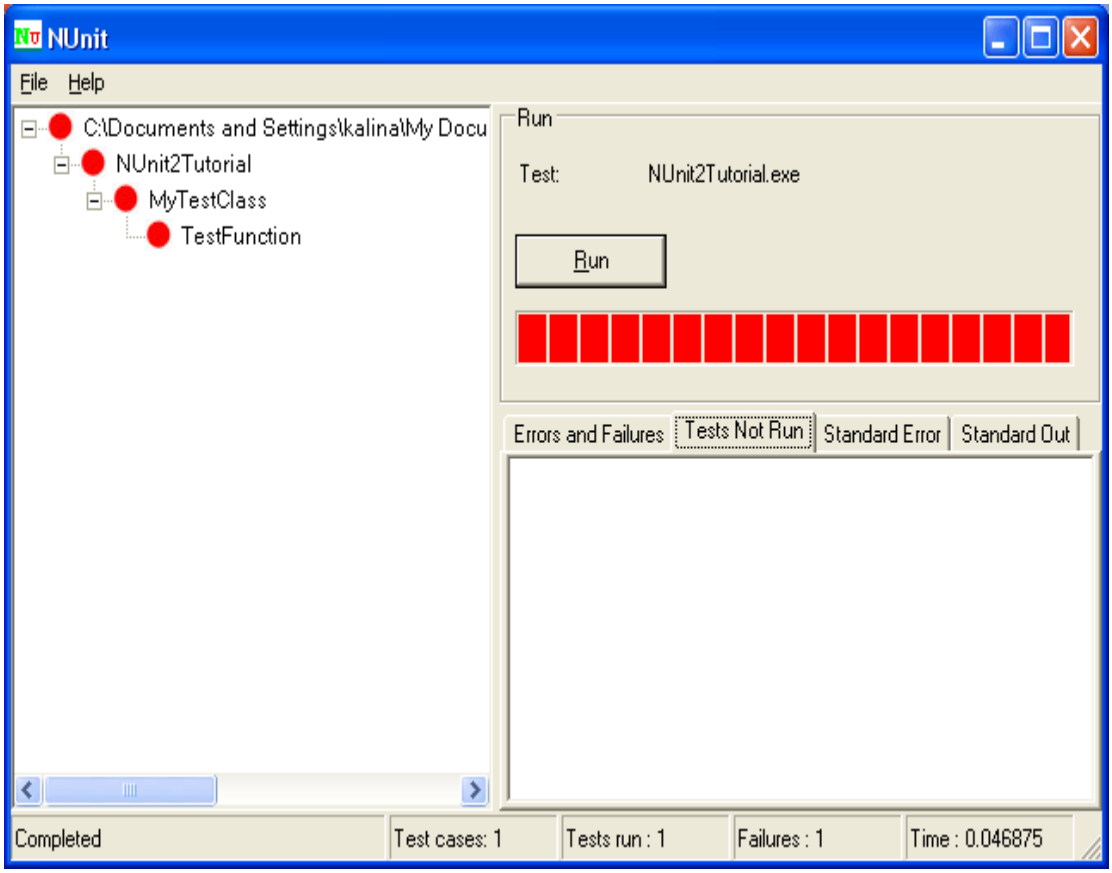
Step 6: Write the following code in the TestFunction.

<<to do>>

```
namespace TestDemo1
{
	[TestFixture]
	public class MyTestClass
	{
		[Test]
		public void TestFunction()
		{
			Assert.AreEqual(1==2);
		}
	}
}
```

Step 7: << to do>>Compile your project with this new code and switch to NUnit-GUI. You will

notice that NUnit has noticed that the library has changed and it has reloaded it for you. Now click run again.



Step 8: <<to do>>Fix this broken test. You would simply change the assertion and go back to NUnit-GUI and hit the run button.

Lab 1 B: Problem statement

Suppose we are writing a bank application and we have a basic domain class – Account. Account supports operations to deposit, withdraw, and transfer funds.
Write the test cases for the above mentioned functions.

Solution:
Step 1: Create a new C# project of Library type.

Step 2: Create the Account class as follows
<< to do>>

```
namespace Bank
{
  public class Account
```



```
  {
    private float balance;
    public void Deposit(float amount)
    {
      balance+=amount;
    }

    public void Withdraw(float amount)
    {
      balance-=amount;
    }
    public void TransferFunds(Account destination, float amount)
    {
    }

    public float Balance
    {
      get{ return balance;}
    }
  }
}
```

Step 3: Create the New project to write Test Class. Write a Test for this class.

<< to do>> The method to be tested is TransferFunds()

```
namespace Bank
{
  using NUnit.Framework;

  [TestFixture]
  public class AccountTest
  {
    [Test]
    public void TransferFunds()
    {
      Account source = new Account();
      source.Deposit(200.00F);
      Account destination = new Account();
      destination.Deposit(150.00F);

      source.TransferFunds(destination, 100.00F);
      Assert.AreEqual(250.00F, destination.Balance);
      Assert.AreEqual(100.00F, source.Balance);
    }
  }
}
```

Step 4: Compile the code & generate a DLL.

Step 5: Start NUnit & Select the TestCase DLL created above.

Run the Test.

The Test should fail. The test has failed because we have not implemented the TransferFunds method yet.

Step 6: Write the code for the TransferFunds Function
<< to do>>

```
public void TransferFunds(Account destination, float amount)
{
        destination.Deposit(amount);
        Withdraw(amount);
}
```

Step 7: Recompile the code & Run the Test again. The test should pass now.
<<to do>>

Step 8: Add the MinimumBalance property to the Account class. It should be readonly.
<< to do>>

Step 9: We will use an exception to indicate an overdraft: Add a new class as follows:

```
namespace Bank
{
  using System;
  public class InsufficientFundsException : ApplicationException
  {
  }
}
```

Step 10: Write a Test method to ensure that the function should throw exception of certain type.

```
[Test]
[ExpectedException(typeof(InsufficientFundsException))]
public void TransferWithInsufficientFunds()
{
        Account source = new Account();
        source.Deposit(200.00F);
        Account destination = new Account();
        destination.Deposit(150.00F);
        source.TransferFunds(destination, 300.00F);
}
```

Note: This test method in addition to [Test] attribute has an [ExpectedException] attribute associated with it – this is the way to indicate that the test code is expecting an exception of a certain type; if such an exception is not thrown during the execution – the test will fail.

Step 11: Compile the code & Run the test.
The Test should fail.

Step 12: Fix the Account code.

```
public void TransferFunds(Account destination, float amount)
{
        destination.Deposit(amount);
        if(balance-amount<minimumBalance)
                throw new InsufficientFundsException();
        Withdraw(amount);
}
```

Step 13: Compile the code & Run the test.
The Test should pass.

Step 14: The code we've just written we can see that the bank may be losing money on every unsuccessful funds Transfer operation. Write a test to confirm our suspicions. Add this test method:

```
[Test]
public void TransferWithInsufficientFundsAtomicity()
{
        Account source = new Account();
        source.Deposit(200.00F);
        Account destination = new Account();
        destination.Deposit(150.00F);
        try
        {
                source.TransferFunds(destination, 300.00F);
        }
        catch(InsufficientFundsException expected)
        {
        }

        Assert.AreEqual(200.00F,source.Balance);
        Assert.AreEqual(150.00F,destination.Balance);
}
```

Step 15: Compile the code & Run the Test.
The test should fail

Step 16: Write the Correct code to fix the Error.

```
public void TransferFunds(Account destination, float amount)
{
        if(balance-amount<minimumBalance)
                throw new InsufficientFundsException();
        destination.Deposit(amount);
        Withdraw(amount);
}
```


Step 17: To temporarily ignore the test, add the following attribute to your test method

```
[Test]
[Ignore("Decide how to implement transaction management")]
public void TransferWithInsufficientFundsAtomicity()
{
        // code is the same
}
```

Step 18: Compile & Run the test. You should get a yellow bar. Click on "Tests Not Run" tab to check the reason.

<<to do>>

Step 19: <<to do>> Perform some refactoring on the code. All test methods share a common set of test objects. Extract this initialization code into a setup method and reuse it in all of our tests.

Note that Init method should have the common initialization code, void return type, no parameters, and marked with [SetUp] attribute.

Step 20: Compile & run the test.

Assignment to do:

Following is the user defined simple implementation of Stack. Write the test cases for this class in NUnit to test whether all operations are performed correctly.

```
public class MyStackClass
{
        private ArrayList elements = new ArrayList();
        public bool IsEmpty
        {
           get
           {
              return (elements.Count == 0);
           }
        }
        public void Push(object element)
        {
           elements.Insert(0, element);
        }
        public object Pop()
        {
           object top = Top();
           elements.RemoveAt(0);
           return top;
```



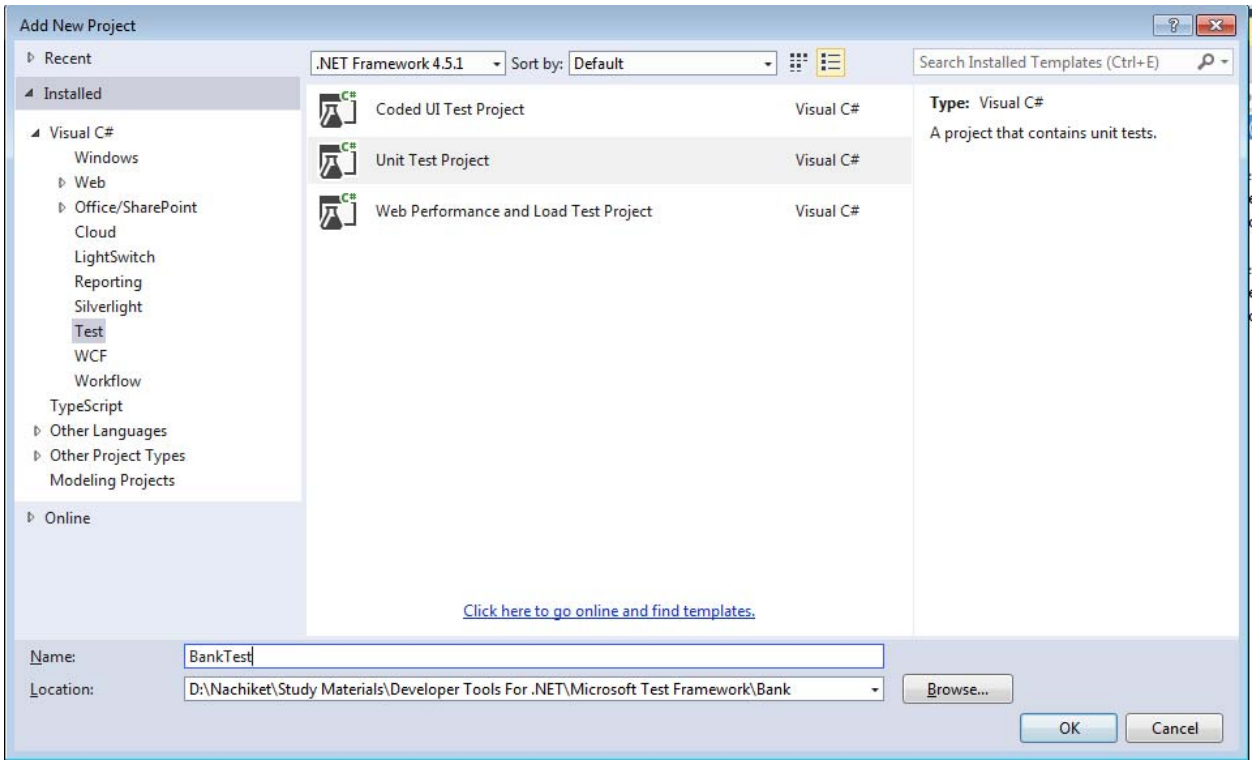
```
        }
        public object Top()
        {
            if(IsEmpty)
                throw new InvalidOperationException("Stack is Empty");

            return elements[0];
        }
}
```

Lab 1 C: Problem statement

The purpose of this lab assignment is to acquaint you with the process of writing test cases using Microsoft Test Framework. In this lab assignment, we will be using Account class created in the lab assignment 1B.

Follow the steps given below to create a unit test project using Microsoft Visual Studio 2013:

1. On the **File** menu, choose **Add**, and then choose **New Project ...**.
2. In the New Project dialog box, expand **Installed**, expand **Visual C#**, and then choose **Test**.
3. From the list of templates, select **Unit Test Project**.
4. In the **Name** box, enter BankTest, and then choose **OK**.
   The **BankTests** project is added to the the **Bank** solution.



5. In the **BankTests** project, add a reference to the **Bank** solution.
   In Solution Explorer, select **References** in the **BankTests** project and then choose **Add Reference...** from the context menu.
6. In the Reference Manager dialog box, expand the **Solution** and then check the **Bank** item.
7. Rename the UnitTest1.cs file in the BankTests project to BankAccountTests.
8. Observe that the following namespace is included by default in the BankAccountTests.cs file:

   using Microsoft.VisualStudio.TestTools.UnitTesting;

   Add the following line in the BankAccountTests.cs file:
   using Bank;

9. Apply the [TestClass] attribute to BankAccountTest class and add the Debit_WithValidAmount_UpdatesBalance() method to the class. After adding this method, the BankAccountTest.cs class should look like this:


```
namespace BankTest
{
  [TestClass]
  public class BankAccountTests
  {
     [TestMethod]
     public void Debit_WithValidAmount_UpdatesBalance()
     {
       // arrange
       double beginningBalance = 11.99;
       double debitAmount = 4.55;
       double expected = 7.44;
       BankAccount account = new BankAccount("Mr. Bryan
                             Walton", beginningBalance);

       // act
       account.Debit(debitAmount);

       // assert
       double actual = account.Balance;
       Assert.AreEqual(expected, actual, 0.001, "Account not
                       Debited correctly");
     }
   }
}
```

Build and Run the Test:

To build and run the test

1. On the **Build** menu, choose **Build Solution**.
   If there are no errors, the UnitTestExplorer window appears with **Debit_WithValidAmount_UpdatesBalance** listed in the **Not Run Tests**group. If Test Explorer does not appear after a successful build, choose **Test** on the menu, then choose **Windows**, and then choose **Test Explorer**.
2. Choose **Run All** to run the test. As the test is running the status bar at the top of the window is animated. At the end of the test run, the bar turns green if all the test methods pass, or red if any of the tests fail.
3. Observe that in Test Explorer, the red/green bar has turned green, and the test is moved to the **Passed Tests** group.

Lab 2: Using NCover to Test Code Coverage

Goals	• Understand working of NCover to Test the code coverage • Understand the steps involved in checking the code coverage.
Time	90 Minutes
Lab Setup	VS.NET 2013, NUnit 2.x Framework, NCover 1.x & Testdriven.NET

Lab 2 A. Problem statement

Write the production code & check the same for code coverage.

Solution:

Step 1: Create your project of Library Type and add methods to it.

```
namespace NCoverDemo.Sample
{
   public class Calculation
   {
      public int AddNumbers(int firstNumber, int secondNumber)
      {
         return firstNumber + secondNumber;
      }

     public int MultiplyNumbers(int firstNumber, int secondNumber)
      {
         return firstNumber + secondNumber;
      }
   }
}
```

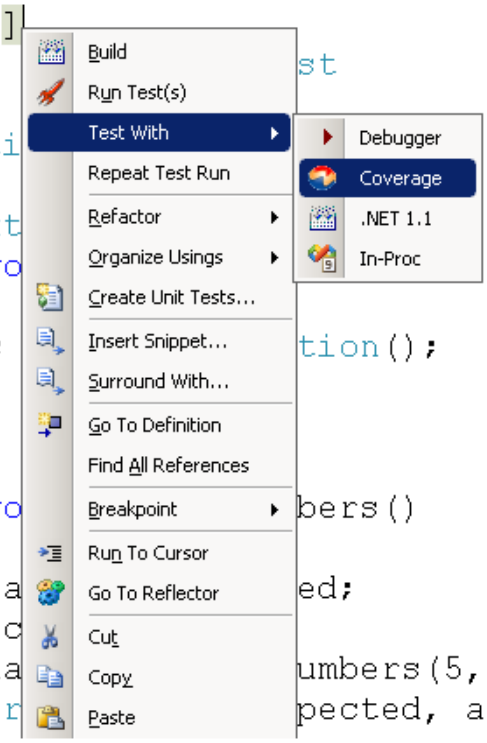

Step 2: Create another project in the same solution of Library type and add test cases to it

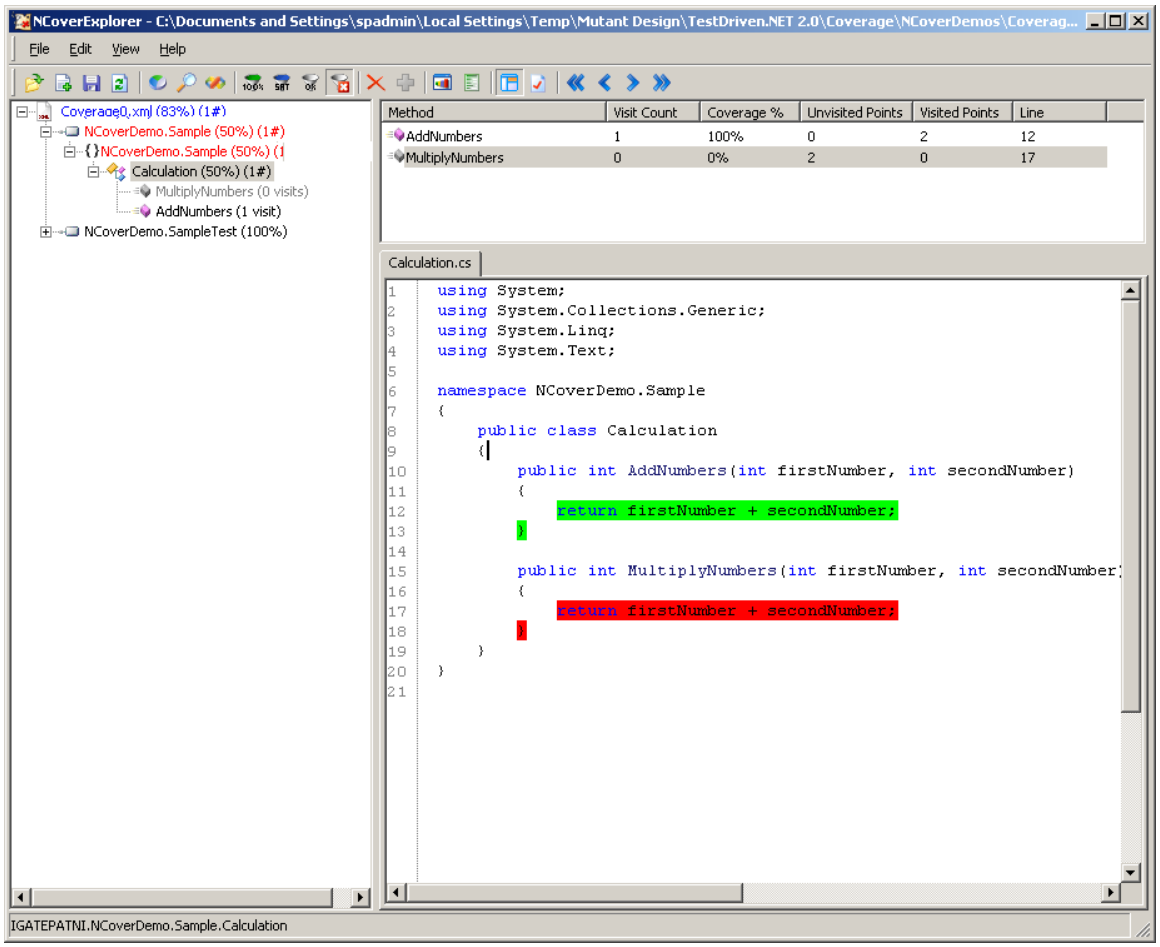
```
namespace NCoverDemo.SampleTest
{
    [TestFixture]
    public class CalculationTest
    {
        Calculation calc = null;

        [TestFixtureSetUp]
        public void Setup()
        {
            calc = new Calculation();
        }
```

```
        [Test]
        public void TestAddNumbers()
        {
            int actual, expected;
            expected = 11;
            actual = calc.AddNumbers(5, 6);
            Assert.AreEqual(expected, actual);

        }
        [TestFixtureTearDown]
        public void TearDown()
        {
            calc = null;
        }
    }
}
```

Step 3: Right click the TestFixture attribute and test for code coverage using NCover

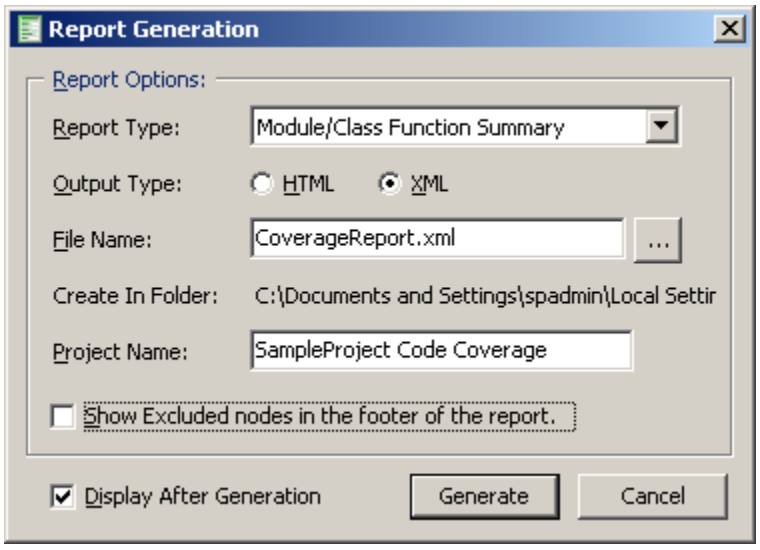

```
namespace NCoverDemo.SampleTest
{
    [TestFixture]
    public class                                    st
    {
        Calculati

        [TestFixt
        public vo
        {
            calc                                    tion();
        }

        [Test]
        public vo                                   bers()
        {
            int a                                   ed;
            expec
            actua                                   umbers(5, 6);
            Asser                                   pected, actual);
        }
```

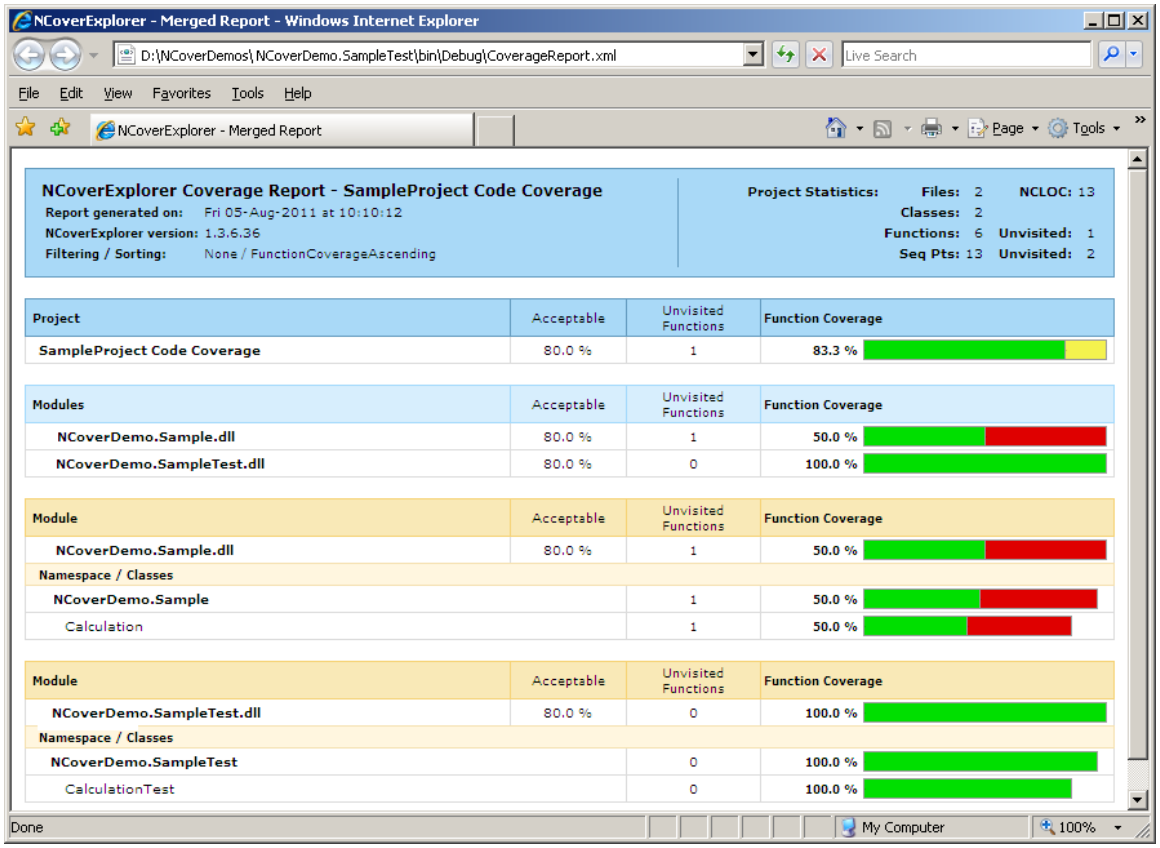




NCoverExplorer shows only 50% of code coverage,highlighting the visited code in green background and unvisited code in red background

Step 4: To create the code coverage report Press F6





<<TODO> ADD the test case for MultiplyNumbers method and generate a code coverage report achieving 100% code coverage

Lab 3: Using Log4Net for logging service

Goals	• Understand working of Log4Net logging service • Understand the steps involved in creating logs using Log4Net.
Time	60 Minutes
Lab Setup	VS.NET 2013, Log4net 1.x

Lab 3 A. Problem statement

Write the production code & add logging service to it.

Solution:

Step 1: Add a reference of Log4net.dll to the project

Step 2: In the App.config file, under Configuration->Configsections, add the following section

```
<section name="log4net" type="log4net.Config.Log4NetConfigurationSectionHandler, log4net" />
```

Step 3: In app.config, add a new section "<log4net>". This section will contain all the settings related to the Log4net configuration.

Step 4: In app.config, under "log4net" section, add the required appender (output target).

```
<appender name="ConsoleAppender" type="log4net.Appender.ConsoleAppender" >
        <layout type="log4net.Layout.PatternLayout">
<param name="ConversionPattern" value="%p - %d{dd-MM-yyyy hh:mm:ss tt} – %m%n"/>
        </layout>
</appender>
```

```
<logger name="ConsoleLogger">
    <level value="ALL" />
    <appender-ref ref="ConsoleAppender" />
  </logger>
```

Step 5: In the app.config file, under "log4net" section, for each appender, add logger and the level (DEBUG, INFO …)

Step 6: To log any information/error/warning, call the appropriate method in the following manner:

```
namespace Log4NetApp.Demo01
{
   public class Sample
   {
      private static readonly ILog log =
LogManager.GetLogger("ConsoleLogger");

      static void Main(string[] args)
      {
         //This calls the default configurator for log4net
         XmlConfigurator.Configure();

         log.Info("Entering application.");

         for (int counter = 1; counter <= 10; counter++)
         {
            log.DebugFormat("Inside of the loop (Counter = {0})", counter);
         }

         try
         {
            throw new NotImplementedException("Testing log4net");

         }
         catch (NotImplementedException ex)
         {
            log.Fatal(ex.Message);
         }

         log.Info("Exiting application.");
      }
   }
}
```

<<todo>
Change the level, in such a way that the console appender should log only Fatal messages

