Summary of Developer Tools for .Net

NUnit

Unit Testing

A unit test is a piece of code written by a developer that exercises a very small, specific area of functionality of the code being tested

Why Unit Testing?

It will make your designs better and drastically reduce the amount of time you spend debugging. To avoid having to compile and run the entire application just to check a single function.

When to do Unit Testing?

Before you start to write code. As you write code. Any other time you can.

Advantages of Unit Testing

Unit testing helps eliminate uncertainty in the pieces themselves and can be used in a bottom-up testing style approach. By testing the parts of a program first and then testing the sum of its parts, integration testing becomes much easier. Unit testing provides a sort of "living document". Clients and other developers looking to learn how to use the class can look at the unit tests to determine how to use the class to fit their needs and gain a basic understanding of the API.

Limitations of Unit Testing

Unit-testing will not catch every error in the program. By definition, it only tests the functionality of the units themselves. Therefore, it will not catch integration errors, performance problems and any other system-wide issues. In addition, it may not be easy to anticipate all special cases of input the program unit under study may receive in reality. Unit testing is only effective if it is used in conjunction with other software testing activities.

NUnit

NUnit is a unit-testing framework for all .Net languages. NUnit brings xUnit to all .NET languages.

Steps for Creating a Test

- Create a separate class library or simply create a new class in the same file as your production code
- Create the tests by using attributes to mark tests
- Write the test in VB or C#
- Use asserts in order to claim that certain things are true, are equal, and so forth

TextFixture Attribute

The TestFixture attribute marks a class as containing tests. Class must reference NUnit.Framework.DLL

Test Attribute

The Test Attribute marks a method as a test. The method must be:

- Public
- Parameterless
- Void in C#, a Sub in VB .NET

Assert Class

The Assert class contains several 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

How to Run a Test?

- 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

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

The ExpectedException Attribute

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.

SetUp and TearDown

SetUp and TearDown are used to mark methods that should be called before and after each test. You may have only one SetUp and one TearDown. Used to reinitialize variables or objects for each test. Tests should not be dependent on each other.

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.

Ignore Attribute

The Ignore attribute is an optional attribute that can be added to a unit test method. This attribute instructs the unit test engine to ignore the associated method. Requires a string indicating the reason for ignoring the test to be provided.

TestClass Attribute

The class which contains the methods used for unit testing is marked with an attribute called TestClassAttribute.

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.

TestInitialize Attribute

Each test case might require a common initialization, like opening a data connection or simply instantiating an object. Such code must not be placed in each test case, but in a separate method. The TestInitialize attribute is used to mark the methods to run before any test. Used to allocate or configure any resources needed by all or many tests in a test class.

TestCleanUp Attribute

This attribute is used to run code after each test has completed. Used to deallocate resources used.

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).

Test Manager Window

Like Test View but can organize tests. Used to show the list of tests, the list of loaded tests, and the list of unorganized tests.

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).

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.

Code Coverage

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

NCover works by monitoring the applications execution using the CLR Profiler. NCover provides a powerful combination with NUnit.

Static Code Analysis

The analysis of computer software that is performed without actually executing programs built from that software.

FxCop

FxCop is an application that analyzes managed code assemblies and reports information about the assemblies, such as possible design, localization, performance, and security improvements. Enforces adherence to .NET Framework Design Guidelines. Internal tool developed by Microsoft to fix common design errors during .NET 1.0 development. 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.

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

FxCop Projects and Targets

In FxCop a "Project" is an FxCop project

- It is not a Visual Studio Project
- It describes the target, rules and exclusions for any given analysis

In FxCop the assembly to be analyzed is called a "target"

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

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 4.7 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.

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
- 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

Logging

Logging is writing the state of a program at various stages of its execution to some repository such as a log file. By logging, explanatory statements can be sent to text file, console, or any other repository. Using logging, a reliable monitoring and debugging solution can be achieved.

Log4Net

Log4Net is an open source logging API for .NET from Apache Software Foundation. log4net is a tool to help the programmer output log statements to a variety of output targets. With log4net it is possible to enable logging at runtime without modifying the application binary.

Features of Log4Net

Support for multiple frameworks

- .NET Framework 1.1, 2.0, Mono

Output to multiple logging targets

- Database, Terminal window, ASP trace context, Applications window Console, Window event log, File System etc.

Hierarchical logging architecture

Dynamic XML Configuration

Components of Log4Net

Log4net is built using the layered approach, with the following components inside of the framework

- Logger, Appender, & Layout

Users can extend these basic classes to create their own loggers, appender, and layouts