NUnit 101 - Beginning TDD in .NET

by Clark Anderson

I have been following Test Driven Development, TDD, for several years. The Open Source tools, I have followed most closely, are JUnit, VBUnit and NUnit.

JUnit, from http://www.junit.org, is designed for JAVA Unit Testing.

VBUnit3, from http://www.vbunit.org, is designed for Visual Basic, Pre .NET.
I have used VBUnit3 to greatly benefit several VB6 projects.

NUnit, from http://www.nunit.org, is a unit-testing framework for all of the .NET languages. I have, recently, been learning how to use NUnit with Visual Studio 2005.

I will share my learning curve with NUnit and Visual Studio:

I started by installing Visual Studio 2005 then downloading and installing NUnit-2.2.8 from http://www.nunit.org .

In this process, I have had a lot of help from the folks at
nunit-users@lists.sourceforge.net
https://lists.sourceforge.net/lists/listinfo/nunit-users

For these examples, my 'product' or 'target' application is Survey05, and my development base directory is D:\Scratch\Lang\VB_NET .

I went through a lot of trial and error and received help from nunit-users, before I arrived at the process I will describe. The VS - NUnit TDD environment, that this process provides, works well. Since I am a .NET green-horn, (especially C#) the naming conventions probably depart from standards.

I will give you this as I continue to learn:

1) Create a new Solution in Visual Studio:
Select menu item: File/New Project...
Other Project Types
Visual studio Solutions
Blank Solution
Name: (e.g.: Survey05)
Location: (e.g.: D:\Scratch\Lang\VB_NET)
Click [OK]

2) Within that solution, create a new product/target Project:
Select menu item: File/Add/New Project...
Visual Basic (or Visual C#)
Windows
Windows Application Template
Name: (e.g.: Survey05)
Location: (e.g.: D:\Scratch\Lang\VB_NET\Survey05)
Click [OK]

I prefer the use of one or more code Modules to contain some reusable Function Procedures.

3) In the product/target Project, add a code Module:
In the solution Browser, select the product/target Project.
Select menu item: Project/Add Module...
Module Template
Name: (e.g.: StrOps2)
Click [OK]

A 'stub' of planned Function Procedures will be started.

4) In the new Module code view/edit window add some code: (e.g.: VB)

```
Option Explicit On
Public Module StringOps2

  Public Function sSQL(ByVal Parameter As Object) As
String
    sSQL = ""
  End Function

  Public Function sXML(ByVal Parameter As Object) As
String
    sXML = ""
  End Function

End Module
```

As you can see, both the Module and the Function are 'Public'. This is critical so NUnit can find and test the Function. The bare minimum Function provides for the planned input parameters and provides some result in the expected data type.

5) In the product/target Project, set Build output path:
Select menu item: Project/Survey05 Properties...
Compile tab
Build output path:
Click [Browse...]
Navigate back to the bin folder and select bin\Debug.

I am providing a consistent location for Build output.

6) Also within the solution, create a new ClassLibrary:
Select menu item: File/Add/New Project...
Visual Basic (or Visual C#)
Windows
Class Library Template
Name: (e.g.: testSurvey05)
Location: (e.g.: D:\Scratch\Lang\VB_NET\Survey05)
Click [OK]

7) For the new test ClassLibrary, Add a Reference to: nunit.framework
In the solution Browser, select the new test ClassLibrary.
Select menu item: Project/Add Reference...
.NET tab
Search for and Select nunit.framework
Click [OK]

8) Also in the test ClassLibrary, Add a Reference to: the target/product project
In the solution Browser, select the new test ClassLibrary.
Select menu item: Project/Add Reference...
Projects tab
Select (e.g.: Survey05 D:\Scratch\Lang\VB_NET\Survey05\Survey05)
Click [OK]

In the Properties window for the new test ClassLibrary, you can change the file name. (e.g.: from Class1.vb to testStrOps2.vb)

9) In the Class Library code view/edit window, lets add some test code!
You really know that things are working when IntelliSense matches your expectations.
(e.g.: VB)

```
Option Explicit On
Imports System
Imports Survey05     'Target/Product Application
Imports NUnit.Framework

<TestFixture()> _
Public Class testStrOps2

  <Test()> _
  Public Sub TestsSQL()
    Assert.Fail("Force Fail without using sSQL()")
  End Sub

  <Test()> _
  Public Sub TestsXML()
    Assert.Ignore("Force Ignore testing sXML()")
  End Sub

End Class
```

(e.g.: C#)

```
using System;
using System.Collections.Generic;
using System.Text;
using Survey05;  //Target/Product Application
using NUnit.Framework;

namespace TestSurvey05
{
  [TestFixture]
  public class TestStrOps2
  {
    [Test] public void TestsSQL() {
      Assert.Fail("Force Fail without using sSQL()");
    }
    [Test] public void TestsXML() {
      Assert.Ignore("Force Ignore testing sXML()");
    }
  }
}
```

10) In the test ClassLibrary Project, set Build output path:
In the solution Browser, select the new test ClassLibrary.
Select menu item: Project/TestSurvey05 Properties...
Compile tab
Build output path:
Click [Browse...]
Navigate back to the bin folder and select bin\Debug.

11) Now we run NUnit!
Select menu item: File/Save All
Select menu item: Build/Build Solution
Select menu item: Tools/NUnit

12) The NUnit-GUI form should open!
(Nunit menu item: Tools/Options/Test: [x] Enable VS support)
Select NUnit menu item: File/Open... (Navigate to the test ClassLibrary Project)
(e.g.: D:\Scratch\Lang\VB_NET\Survey05\testSurvey05\testSurvey05.vbproj)

13) The NUnit Tests tree should populate.
Click [Run]

We know our test is running with all of the red circles on the left pane and the big red meter bar in the right pane. TestsXML will have a yellow circle because its test was ignored.

14) It is time to add some real test code:
(e.g.: VB)

```
Option Explicit On
Imports System
Imports Survey05     'Target/Product Application
Imports NUnit.Framework

<TestFixture()> _
Public Class testStrOps2
    <Test()> _
    Public Sub TestsSQL()
        Assert.AreEqual("''", sSQL(DBNull.Value),
"Null Parameter")
        Assert.AreEqual("''", sSQL(""), "''
Parameter")
        Assert.AreEqual("'25'", sSQL(25), "25
Parameter")
        Assert.AreEqual("'True'", sSQL(True), "True
Parameter")
        Assert.AreEqual("'False'", sSQL(False),
"False Parameter")
        Assert.AreEqual("'Smith'",
StringOps2.sSQL("Smith"), "'Smith' Parameter")
        Assert.AreEqual("""O'Maley""",
sSQL("O'Maley"), "'O'Maley' Parameter")
    End Sub
    <Test()> _
    Public Sub TestsXML()
    '    Assert.Ignore("Force Ignore testing sXML()")
    End Sub
End Class
```

(e.g.: C#)

```
using System;
using System.Collections.Generic;
using System.Text;
using Survey05;      //Target/Product Application
using NUnit.Framework;

namespace TestSurvey05
{
  [TestFixture]
  public class TestStrOps2
  {
    [Test] public void TestsSQL(){
      Assert.AreEqual("''",
StringOps2.sSQL(DBNull.Value), "Null Parameter");
      Assert.AreEqual("''", StringOps2.sSQL(""), "''
Parameter");
      Assert.AreEqual("'25'", StringOps2.sSQL(25),
"25 Parameter");
      Assert.AreEqual("'True'",
StringOps2.sSQL(true), "True Parameter");
      Assert.AreEqual("'False'",
StringOps2.sSQL(false), "False Parameter");
      Assert.AreEqual("'Smith'",
StringOps2.sSQL("Smith"), "'Smith' Parameter");
      Assert.AreEqual(@"""O'Maley""",
StringOps2.sSQL("O'Maley"), "'O'Maley' Parameter");
}
    [Test] public void TestsXML() {
    //  Assert.Ignore("Force Ignore testing sXML()");
    }
  }
}
```

The first parameter in each Assert statement is the expected value. The second is the actual value, and the third is the message.
It is important that each message in a test is unique to distinguish one Assert result from another.

You may fully specify the target Function name (e.g.: Survey05.StringOps2.sSQL()) in VB or C#.
In C# it is required to specify the Module name as well as Function name (e.g.: StringOps2.sSQL()).

15) Save changes and rebuild:
Select menu item: File/Save All
Select menu item: Build/Rebuild Solution

16) Refocusing on the NUnit-GUI form, the Tests tree should have repopulated.
Click [Run]

Oh Well, it still failed. Our TestsSQL() Assert statements define the requirements for the finished sSQL() Function Procedure (Test Driven Development).
In the left pane, TestsXML has a green circle because the Assert.Ignore() statement has been 'commented out'. That 'empty' test passes.

The Error and Failures pane on the middle right gives details of each failure (Message, Expected value and Actual value). When a failure is selected (mouse click), more specifics, including the test code line number, are displayed in the lower right pane.

17) It is time to develop some code for the sSQL Function that will pass the test:
In the Module code view/edit window, replace original 'stub' code with:
(e.g.: VB)

```
Option Explicit On
Public Module StringOps2
  Public Function sSQL(ByVal Parameter As Object) As
String
    Dim sParam As String
    Const cA As String = "'"
    Const cQ As String = """"
    Const cTQ As String = """""""""
    sParam = Trim$(CStr("" & Parameter))
    If InStr(sParam, cA) > 0 Then
      If InStr(sParam, cQ) > 0 Then
        sSQL = cTQ & sParam & cTQ
      Else
        sSQL = cQ & sParam & cQ
      End If
    Else
      sSQL = cA & sParam & cA
    End If
  End Function
  Public Function sXML(ByVal Parameter As Object) As
String
    sXML = ""
  End Function
End Module
```

18) Save changes and rebuild:
Select menu item: File/Save All
Select menu item: Build/Rebuild Solution

19) Refocusing on the NUnit-GUI form, the Tests tree should have repopulated.
Click [Run]

NUnit results pass! It is all green circles in the Tests tree and a nice big green bar.

This is a tiny example of Test Driven Development.

Ideally, the Assert statement collection tests each possibility for input parameters and each logic flow path within the target Function Procedure. This is referred to as Coverage.

Creating NUnit test :
- Before development is ideal!
- Before major changes is quite beneficial.
- Any time for critical code blocks can be lifesaver!

This unit testing code is not a built-in part of the shippable product because it is in a separate project within the solution. It is just as available and takes some of the burden off regression testing.

There are more than a dozen Assert methods available for use with NUnit.

Learn more:

http://www.nunit.org
https://lists.sourceforge.net/lists/listinfo/nunit-users
http://www.testdriven.net/ (.NET 2.0 Beta)
(Looks promising for better integration!)