Unit Test Visualisation
Unit test coding convention guide

Author: Keagan Phillips
Team: Group 4
Course: ELEN7046 - Software Technologies and Techniques
Date Submitted: 25 June 2012
Source & Documentation: https://github.com/KeaganPhillips/Wit-Group-4-project/tree/master/Documentation

Abstract

This document describes the coding convention developers need to adhere to that enable the 'Unit Test Visualisation' system to extract test metadata.

Contents

1 Introduction

1.1 Overview

The Unit Test Visualisation project extracts unit test metadata and present that data to the user on an HTML web front end as well as a PDF document. In order to extract the unit test metadata, developers must conform to a strict unit test coding standard. This convention is important because it provides the system to with an predictable, predefined framework from which unit test metadata can be extracted.

1.2 Document Scope

The rest of this document outlines this convention and discusses the technical reasoning around it.

1.3 Assumptions Made

This document assumes that the developer will be using the 'Visual Studio Unit Testing Framework'[3] as a unit test testing framework, and that the developer is familiar with the product.

2 Organising Tests

2.1 Folder structure

Figure 1 depicts the folder structure that tests should conform to. As depicted, all classes for which there are test written for should have their own folders and must be contained within the parent ClassesUnderTest folder. Each class folder (e.g. Account) in turn may contain one or more "Test" folders, as a class can have multiple tests. A test on the other hand may have one or more scenarios. The scenario source files is where the actual automated test code will be written.

This structure makes it very easy for developers to arrange, find and maintain unit tests.

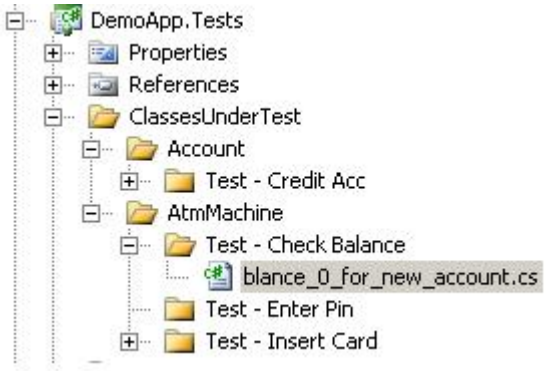

Figure 1:

3 Test Scenarios

3.1 Given When Then

Test scenarios are written using the 'Given / When / Then'[1] convention. Each scenario must implement the IScenario interface. Figure 2 shows a code snippet of this interface.

3.2 IScenario Interface

By implementing IScenario, developers are forced to explicitly provide 'Given' and 'When' descriptions for the test scenario in question, where 'Given' is an description of the initial context (initial state) of the class and it's dependencies and 'When' is an description of the event that occurs. Developer implement the ScenarioDescription property to provide a meaningful explanation of the test scenario. The ClassUnderTest property returns the class type for which the scenario is applicable.

The IScenario interface also provides Given and When methods where developers implement the code that sets up the initial context (i.e. given) and execute the event (i.e. when).

The developer must decorate their test method with at 'Then' C# attribute as shown in figure 3 to provide a meaningful description of the expected state of the class and dependencies after the event occurred. The reason for using attributes in this case is because a test may have more than one test criteria

```
public interface IScenario
{
    Type ClassUnderTest { get; }

    string SecnarionDescription { get; }
    string GivenDescription { get; }
    string WhenDescription { get; }

    void Given();
    void When();
}
```

Figure 2:

```
[TestMethod]
[ThenDescription(@"ATM status is 'Card Inserted'")]
public void atm_status_is_card_inserted()
{
    Assert.IsTrue(_atmMachine.CardInserted);
}
```

Figure 3:

3.3 The TestName Attribute

Each scenario must be decorated with the TestName C# Attribute as shown in Figure 4. The framework uses this attribute to determine the test the scenario belongs to.

3.4 The TestHelper

Each scenario must call 'TestHelper.SetupTest(this);' (see Figure 4) as part of the test setup. This helper method will call the IScenario.Given() and IScenario.When() methods for the current class to ensure the the initial context is setup and the event is fired.

4 Conclusion

By following these conversions, the framework should extract the unit test metadata to be presented to the user on the front end.


[TestClass]
[TestName("Debit/Credit Account Test")]
public class can_credit_acc:IScenario
{
[TestInitialize]
public void Setup()
{
TestHelper.SetupTest(this);
}
private DemoApp.Account _account;
public Type ClassUnderTest
{
get { return typeof(DemoApp.Account); }
}
public string SecnarionDescription
{
get { return "Can credit an Account"; }
}
Given
When
Then
}

Figure 4:

References

[1] Structure your test using Given/When/Then; Tested Objects 1.0 Users' Guide; http://testedobjects.sourceforge.net/m2-site/main/documentation/docbkx/html/user-guide/ch03s04.html

[2] Attributes (C# Programming Guide); MSDN

http://msdn.microsoft.com/en-us/library/z0w1kczw%28v=vs.80%29.aspx

[3] Visual Studio Unit Testing Framework; wikipedia

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