Create Unit Tests

Step by step Exercises

Hans-Petter Halvorsen, M.Sc.
Levels of Testing

Unit Testing: Test each part independently and isolated

Integration Testing: Make sure that different pieces work together. Test the Interfaces between the different pieces. Interaction with other systems (Hardware, OS, etc.)

Regression Testing: Test that it still works after a change in the code

System Testing: Test the whole system
What are Unit Tests?

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

• Unit Testing (or component testing) refers to tests that verify the functionality of a specific section of code, usually at the function level.
• In an object-oriented environment, this is usually at the class and methods level.
• Unit Tests are typically written by the developers as part of the programming
• It is also known as component testing.
• Typically Automatically executed (e.g., Visual Studio and Team Foundation Server have built-in functionality for Unit Testing)
Test Driven Development (TDD)

• Coding and Testing are done in parallel
• The Tests are normally written before the Code
• Introduced as part of eXreme Programming (XP) (an Agile method)
• Unit Tests are important part of Software Development today – either you are using TDD or not
Unit Tests Frameworks

Unit Tests Framework are usually integrated with the IDE

- **Visual Studio Unit Test Framework.** Unit Tests are built into Visual Studio (no additional installation needed)

Others:

- **JUnit** (Java)
  - JUnit is a unit testing framework for the Java programming language.

- **NUnit** (.NET)
  - NUnit is an open source unit testing framework for Microsoft .NET. It serves the same purpose as JUnit does in the Java world

- **PHPUnit** (PHP)
- **LabVIEW Unit Test Framework Toolkit**
- **etc.**

All of them work in the same manner – but we will use the Visual Studio Unit Test Framework
Exercises

Visual Studio:
1. Create Unit Tests in Visual Studio
2. Code Coverage

Project:
4. Create Unit Tests for your Code in your Project
Create Unit Tests in Visual Studio

• Visual Studio have built-in features for Unit Testing
• In this example we will create a simple Class Library
• Then we will create a Test Class in order to test our code (include a Test Project in our Solution)

Create the Code you want to Test

Create New Project

Bank

Class Library

Step by step Example
using System;
namespace BankAccountNS
{
    public class BankAccount
    {
        private string m_customerName;
        private double m_balance;

        public BankAccount(string customerName, double balance)
        {
            m_customerName = customerName;
            m_balance = balance;
        }

        public double Balance
        {
            get { return m_balance; }
        }

        public void Debit(double amount)
        {
            if (amount > m_balance)
            {
                throw new ArgumentOutOfRangeException("amount");
            }
            if (amount < 0)
            {
                throw new ArgumentOutOfRangeException("amount");
            }
            m_balance += amount;
        }
    }
}
Add New Project: Unit Test Project

Create a proper Name for your Project
Your Solution should now have 2 Projects

Add Reference to the Code under Test
Create Test Class “BankAccountTests.cs”

We need a Test Class for verifying the “BankAccount” class. We can use the UnitTest1.cs that was generated by the project template, but we should give the file and class more descriptive names. We can do that in one step by renaming the file in Solution Explorer.

```csharp
using System;
using BankAccountNS;

namespace BankTest
{
    [TestClass]
    public class BankAccountTests
    {
        [TestMethod]
        public void TestMethod1()
        {
        }
    }
}
```

Reference to Test Framework

Make sure to add reference to the code under test

Note! The Test Class should be labeled `[TestClass]` and the Test Methods need to be labeled `[TestMethod]`
Test Method Requirements

A test method must meet the following requirements:

• The method must be decorated with the `[TestMethod]` attribute.
• The method must return void.
• The method cannot have parameters.
The basic concept in Unit Testing is to Compare the results when running the Methods with some Input Data ("Actual") with some Known Results ("Expected")

Example:

```
...  
Assert.AreEqual(expected, actual, 0.001, "Test failed because..." );
```
PHILOSOPHISING GEEKS

SOMETIMES I ASK MYSELF: "BUT WHAT IF THIS TEST FAILS?"

assert(true);
Create the first Test Method

We will write unit test methods to verify the behavior of the “Debit” method of the “BankAccount” class.

Create the following Method in the “BankAccountTests.cs” Class:

Debit_WithValidAmount_UpdatesBalance()

```csharp
[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");
}
```

Expected = 
Balance - debitAmount
= 11.99 - 4.55 = 7.44
Running the Unit Test using the “Test Explorer”

If Test Explorer does not appear after a successful build, choose Test on the menu, then choose Windows, and then choose Test Explorer.

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.

In this case, the test does fail. The test method is moved to the Failed Tests group.

Select the method in Test Explorer to view the details at the bottom of the window.
Can you find the Bug based on the Test Results?
Analyzing the Test Results

- The test result contains a message that describes the failure.
- For the AreEquals method, message displays you what was expected (the (Expected<XXX> parameter) and what was actually received (the Actual<YYY> parameter).
- We were expecting the balance to decline from the beginning balance, but instead it has increased by the amount of the withdrawal.
- A reexamination of the Debit code shows that the unit test has succeeded in finding a bug. The amount of the withdrawal is added to the account balance when it should be subtracted.
Correction the Bug

m_balance += amount;  ->  m_balance -= amount;

Rerun the Test

In Test Explorer, choose **Run All** to rerun the test. The red/green bar turns green, and the test is moved to the **Passed Tests** group.
You are finished with the Exercise
Code Coverage

- Code coverage is a measure used in software testing. It describes the degree to which the source code of a program has been tested.
- Depending on the input arguments, different parts of the code will be executed. Unit Tests should be written to cover all parts of the code.
Run Code Coverage

• We see that our test only cover about 43% of the code in our “Debit” Method.
• This is because the Method contains if sentences
• We should improve the Unit Testing so that we cover more of the code.

=> Try to Improve the Unit Testing so it covers 100% of the code in the Debit Method
You are finished with the Exercise
More about Unit Testing in Visual Studio

If you want to learn more, visit the following:

Verifying Code by Using Unit Tests:


The UnitTesting namespace, which provides attributes, exceptions, asserts, and other classes that support unit testing.

The UnitTesting.Web namespace, which extends the UnitTesting namespace by providing support for ASP.NET and Web service unit tests.
Project Assignment

• Create Unit Tests for some parts of your Code in your Project
You are finished with the Exercise
Poenget er ikke å lure testene, men å sørge for at koden er feilfri!
Unit Tests – Best Practice

• A Unit Test must only do one thing
• Unit Test must run independently
• Unit Tests must not be depend on the environment
• Test Functionality rather than implementation
• Test public behavior; private behavior relates to implementation details
• Avoid testing UI components
• Unit Tests must be easy to read and understand
• Create rules that make sure you need to run Unit Tests (and they need to pass) before you are allowed to Check-in your Code in the Source Code Control System

http://www.uio.no/studier/emner/matnat/ifi/INF5530
References

• Wikipedia: NUnit:  http://en.wikipedia.org/wiki/NUnit
• Course at UiO: Foundations of Software Testing: http://www.uio.no/studier/emner/matnat/ifi/INF5530
Hans-Petter Halvorsen, M.Sc.

University College of Southeast Norway

www.usn.no

E-mail: hans.p.halvorsen@hit.no

Blog: http://home.hit.no/~hansha/