

Effective Unit Testing

By
Philip Japikse
Phil.Japikse@pinnsng.com
MVP, MCSD.Net, MCDBA, CSM
Principal Consultant
Pinnacle Solutions Group



- ▶ Principal Consultant, Pinnacle Solutions Group, Inc.
 - Senior Level consulting company specializing in Architecture and Development of Business Applications and Business Intelligence Solutions
- ▶ Microsoft MVP, MCSD.Net, MCDBA, CSM
- ▶ Enterprise Application Architect
- ▶ Trainer/Mentor
 - Creator of customized training for development teams
- ▶ National and Regional Speaker
- ▶ Director, Cincinnati .NET User's Group

Development Environment

- ▶ IDE = VS 2008 (C#)
 - ReSharper 4.5.xx
- ▶ Unit testing frameworks
 - Gallio/MbUnit 3.0.6.xx
 - JSSpec (JavaScript)
 - T-SQL Test Tool
- ▶ Mocking = Rhino Mocks 3.5, TypeMock
- ▶ Code Coverage = NCover 3
- ▶ Database = SQL Server 2008
- ▶ NHibernate 2.0.1

Unit Testing Motivation

- ▶ Cost of correcting a defect (by phase)¹:
 - *Requirements* = \$139
 - *Design* = \$455
 - *Coding* = \$977
 - Unit testing targets coding → deployment and maintenance
 - *System Testing* = \$7,136
 - *Maintenance* = \$14,102

¹B.Boehm and V. Basili, "Software Defect Reduction Top 10 List", IEEE Computer, January 2001



- ▶ **Unit Testing Definitions**
- ▶ Unit Testing Patterns and Practices
- ▶ Code Demos

Unit Testing Frameworks

- ▶ Tool used to run Unit Tests and provide feedback, either visually or as part of the build process
- ▶ .Net Frameworks (a few)
 - MbUnit, nUnit, xUnit, MSTest, etc
- ▶ JavaScript Frameworks
 - JSSpec
- ▶ SQL Server
 - T-SQL Test Tool



- ▶ A procedure used to validate that individual units of source code are working properly¹.
 - A unit is the smallest testable part of an application.
 - Procedural Programming:
 - An individual program, function, or procedure
 - OOP:
 - Method which may belong to a base/super class, abstract class or derived/child class.

¹Courtesy of Wikipedia

The Law of Unit Testing

- ▶ Each Unit Test is independent of any other unit test
- ▶ Each Unit Test leaves the System Under Test in the same state as before the test was executed
 - Otherwise, tests are not repeatable
 - Integration tests also need to follow this rule

Types of Unit Tests

- ▶ State Testing
 - Easiest tests to write and execute
 - Asserted with value based semantics
 - AreEqual, IsNull, IsTrue, etc
 - Think *Stubs*
- ▶ Behavior-based Testing
 - Verification of Behavior of SUT
 - Achieved through mocking objects
 - Asserted via mocking frameworks
 - `mockObject.VerifyAllExpectations()`
 - Think *Mocks*



- ▶ Object (Class) that provides context and specific resources to run one or more Unit Tests
- ▶ Can inherit from base Test Fixture
- ▶ Provide Specific method attributes for test runner
 - Class level
 - Fixture/FixtureSetUp/TearDown
 - Test Level
 - SetUp/TearDown/Test

Test Fixtures Methods

- ▶ Fixture Methods – fire once per Test Fixture
 - FixtureSetup – before any tests
 - FixtureTearDown – after all tests
- ▶ Test Methods – fire once per test
 - SetUp – before each test
 - TearDown – after each test
- ▶ Test Attributes
 - Row(MbUnit/nUnit Extensions) – allows for seamless data driven testing
 - ExpectedException – to test exception paths
 - Ignore – should this be used?



- ▶ Code Coverage
 - Quantifiable metric through tools
 - NCover, TFS, etc
 - Measures lines of code executed by the Unit Tests
 - For my teams, we target 80%
 - Opinions vary on the amount
 - Regardless of target, this should NOT be a management metric

Use Case Coverage



- ▶ Use Case Coverage
 - Do the tests cover more than just the happy path?
 - Are exceptions and exception handling covered?
 - What about fringe cases?
 - Int.MaxValue, DateTime.MinValue, etc
 - Assert.AreEqual(4, Add(3, 1))
 - Will pass if Add is hardcoded to return 4

Mocks and Stubs¹



▶ Mocks

- Are pre-programmed with expectations
- Used to verify the *behavior* of the system under test
 - Behavior is arranged in the Test

▶ Stubs

- Provide canned answers to calls
- Used to verify *state* of the system under test
 - State (result of calls) is predefined in the Stub

¹<http://martinfowler.com/articles/mocksArentStubs.html>

Why Mock Objects?

- ▶ Conditions that are difficult to reproduce
 - Specific time of day
 - A failing service
- ▶ Objects that are slow to execute or setup
 - Web service calls
 - Databases
- ▶ Objects that are not yet coded
- ▶ Objects that introduce noise that could occlude test results

The “T”s and “D”s

- ▶ “TED” – Test Eventually Development
 - May be only way in certain situations (generated code)
 - Better than nothing, but not a “roadmap” to TDD
- ▶ “TDD” – Test Driven Development
 - Why we’re here 😊
- ▶ “BDD” – Behavior Driven Development
 - TDD “done right”
- ▶ “DDD” – Domain Driven Design
 - An ally of TDD
 - Championed by Eric Evans, definitely worth investigating



- ▶ Unit Testing Definitions
- ▶ Unit Testing Patterns and Practices
- ▶ Code Demos

Solution / Project Structure

- ▶ Every Line Of Business (LOB) project should have a test project named [LOBName]Tests
 - DAL/DALTests
- ▶ Within the test project
 - Match Folder for Folder
 - Every Class in LOB project should have a Test Fixture name [LOBClassName]Tests



- ▶ Test Names should describe:
 - What is being tested
 - The condition being tested
 - The expected result
- ▶ Three schools of thought:
 - Sentence
 - Should_Return_Sum_When_Adding_Two_Integers
 - Should_Throw_When_Adding_With_Negative_Addend
 - Phrase (MethodTested_Action_Result)
 - Add_TwoIntegers_ReturnsResult
 - Add_NegativeInteger_ThrowsException
 - Given, When, Then
 - GivenTwoIntegers_WhenAdding_ThenReturnSum
 - GivenNegativeInteger_WhenAdding_ThenException



- ▶ Unit Tests should follow the Arrange, Act, Assert pattern
 - Arrange
 - Setup system under test
 - Provide necessary data/dependencies
 - Act
 - Exercise the Unit of Code
 - Assert
 - Confirm the expected result
 - Sometimes, this is a non-change

Dependency Injection

- ▶ AKA Inversion of Control
- ▶ Separates Client from instantiation/implementation of the Service
- ▶ Three methods
 - Constructor Injection
 - Setter Injection
 - Interface Injection
- ▶ Used to:
 - Create *seams* in the system under test to enable unit testing
 - Isolate the system under test to provide better testability

Tracking (Test) To-Do's

- ▶ Keep list of tests || work items by workstation
 - Paper works best for me
 - Electronic impedes my rhythm
- ▶ As ideas come up, write them down
- ▶ Tackle them in order of confidence
- ▶ Cross them off when done
- ▶ Remember, Computers can multi-thread, people cannot

Source Code Control

- ▶ Once all tests are green COMMIT (Check In) all code and tests
 - Immediately UPDATE (Get Latest)
 - Rerun all tests
- ▶ If change breaks multiple tests, REVERT
 - Often quicker than chasing the “white rabbit”



- ▶ Automated builds need to also run all unit tests
 - Builds should be run at least twice a day (striving towards Continuous Integration)
 - Failed Test = Failed Build = Doughnuts



- ▶ Unit Testing Definitions
- ▶ Unit Testing Patterns and Practices
- ▶ Code Demos



- ▶ Phil.japikse@pinnsng.com
- ▶ www.pinnsng.com
- ▶ www.japikse.blogspot.com
- ▶ www.twitter.com/skimedic
- ▶ (513) 619-6323