



FISCHER & CONSULTANTS

welcomes you to
the DevFest 2007 Session

**Test-driven Development
with Vulcan.NET and Visual Objects**

Speaker
Michael Fischer

Copyright © 2007
Fischer & Consultants GmbH (F&C)
Martinstrasse 1
44137 Dortmund
www.appfact.com

Kontakt: Michael Fischer
Email mifi@appfact.de
Phone +49 (172) 6801489
Fax +49 (941) 599212011

Contents

- | | |
|---|--|
| 1. Basics | 6. More Testing Methods |
| 2. Choosing your weapon
under .NET:
NUnit vs. VSTS | 7. Integration of TDD into a
Software Development
Process |
| 3. Introduction to NUnit | 8. Questions and
Discussion |
| 4. A Class in the Life of a
Test-Driven Developer | |
| 5. VOUnit | |

Slide 2 of 27

Abstract:

Test, Test, Test ...

Unit Tests with Visual Objects and Vulcan.NET

Test-driven Development (TDD) has become popular with *Agile Software Development* methodologies, for example *Extreme Programming*. It puts writing automated tests before you start the actual coding. The tests drive the design of the component to be developed which leads not only to more solid code but also enforces a clear and comprehensive class design.

Whereas the rest of the software world already have tools like JUnit, NUnit or Visual Studio Team Services to bring a new quality into their code, the VO community has fallen behind due to the lack of an equivalent tool. Vulcan.NET, the first Xbase language for .NET, changes the picture. Using an example, Michael Fischer will do a live demonstration of how test-driven development with Vulcan.NET and NUnit feels in everyday life. He examines the influence this method has on the final results and why you get addicted to „call it a day with a green light shining“.

But what about VO? The good news is that Meinhard Schnoor-Matriciani has written VOUnit for his team at Fischer & Consultants, a simple and straight forward implementation of Unit Tests for Visual Objects. In the last part of his session, Michael Fischer will introduce VOUnit and demonstrate how you can write automated tests with VO today. The sourcecode of this tool will be provided with the conference CD.

1. Basics

Test-Driven Development

Test-Driven Development (TDD) is an **Agile Method** for software development which puts **writing the tests before writing the actual code**.

- à Became popular by Kent Beck's book about Extreme Programming (1999)
- à Design-Principle and part of an overall software development process

Unit Test

Unit tests are executable code fragments which can be run **automatically**. They verify the public interface of a a component or class.

- à Grey-Box-Test
- à Focus on one (**atomar**) statement, **isolated** from each other

Slide 3 of 27

Agile Methods are methods to implement an agile software development process like **Extreme Programming**. Agile software development emphasizes values, principles, communication and code over traditional process models based on written documents. TDD should always be seen in the context of all other methods of a particular agile model (pair programming, refactoring, continuous integration etc.).

A **Grey-Box-Test** has both aspects of a White-Box-Test and a Black-Box-Test. It is *white* because it is written by the same person writing the actual code. It is *black* because it does not know any internals of the tested component (remember: it is written before the actual code).

Integration Tests (like acceptance tests) are not to be confused with unit tests. Integration tests are grouped tests in a particular order that operate on aggregated components (which should have been unit tested before) to check the collaboration of these components. They are not covered in this session.

1. Basics

Test Framework

is a framework for the execution of automated test.

Consists of:

n Classes (and attributes) to program the unit tests

n Test runners
Console and GUI

n IDE-Integration (optional)

Usually, the console runner will be integrated into a build process (MSBuild, Nant, CruiseControl).

Slide 4 of 27

History

- SUnit for Smalltalk is said to be the first xUnit system
- JUnit for Java (Erich Gamma and Kent Beck)
- NUnit for .NET Welt

All three projects are hosted on SourceForge.NET.

Implementation

The implementation of xUnit frameworks in modern languages rely on this language features:

- Attributes (.NET) or Annotations (Java) to mark test cases and test methods
- Dynamic loading of libraries with test classes by the test runner
- Test runner uses reflection to identify and analyse the test cases in the test libraries

1. Basics

Test-Driven Cycle

1. Pick a test to implement.

Keep it simple. Can you implement it now? Is it the next logical step?

2. Write the test.

It will fail.

3. Quickly, make the test green.

4. Clean up your code.

Refactor: Eliminate duplication. Change implementations where necessary. Check code conventions.

Slide 5 of 27

You cannot test everything. Writing no tests at all is not an option either. How do you pick the right tests, the rewarding ones?

Tests are a bet. You bet on something to properly function. If the test proves the opposite, nothing is more important than to make the test pass. Sometimes you bet on the opposite (when you expect an exception). The more you write tests, the more you will learn to write the tests that pay off.

Code Coverage analysis can help you to find the right amount of tests. If they report that only a small fraction of the code is run during the automated tests, you probably do not have enough tests. But code coverage does not tell you about the quality of your test cases.

1. Basics

TDD Advantages

- n Maintenance and total cost of ownership**
Refactoring (changing the implementation) of a component will not break existing functionality.
- n Design Quality**
Developers have to consume their own classes even before they can write them.
- n Code Quality**
The quality of code directly increases with the number of tests.
- n Less side effects**
Developers work on building and running systems most of the time.

Slide 6 of 27

Kent Beck mentions another advantage: developers gain more confidence in their own code.

Another advantage is the documenting function of test code. A good test case with sensefull names for the test methods almost replaces a quick start manual for a component.

1. Basics

TDD Problems

n Existing projects

- à no existing tests à maybe no test framework available
- à no separation of GUI code and functional code

n Not suited for a software novice

„How can I write a test for something that doesn't even exist?“

n Bad tests

You feel secure. You aren't.

n Difficulties to isolate tests

Tests depend on external systems
(Interfaces, Databases, Services)

n Missing tools for database projects

No first class tools available to test server-side business logic written in SQL.

Slide 7 of 27

Existing system without any unit tests, no or no sufficient distinction between GUI and functional code, no test framework available for the language. Does that sound familiar to you? It describes most VO projects. That's why we will cover some of the aspects of introducing TDD to an existing project later on in the chapter about VOUnit.

The Extreme Programming model addresses the problem of software novices having to write tests with another method: **Pair Programming**. Another answer to this problem could be training of **Test Patterns**, like Mark Clifton describes them in his article on codeproject.com.

External systems can be emulated by **Mock-Objects**.

2. NUnit vs. VSTS

NUnit

n (+) Open source, no royalties

n (+) Mature

à Based on other xUnit implementations

à First release in 2002

à Version 2 reimplemented for the .NET framework

n (+) Open

à VS 2003 à VS 2005 à VS Express à Sharp Develop à Mono

n (+) Designed for TDD

n Add-Ons

e.g. NCover (code coverage) or TestDriven.NET (VS integration)

This session is based on version 2.2.5

2. NUnit vs. VSTS

Visual Studio 2005 Team System (VSTS)

n (+) Seamless integration with other Team System functions
IDE integration, code coverage, project management, ...

n (+) Test case generation

n (-) Only Visual Studio 2005 Team Editions, fairly expensive

VS 2005 Professional: 811,21 €

VS 2005 Team Developer: 5.586,21 €

(Zoschke, August 2006)

Slide 9 of 27

Conclusion

Most teams are well equipped with VS Standard or VS Professional and NUnit. The OEM-version of VS shipped with Vulcan.NET is not sufficient, IMHO, because it does not support C# and VB.NET which you need to step through all the examples for .NET technologies available in the internet (e.g. codeproject.com, gotdotnet.com or other portals).

3. Introduction into NUnit

Steps to SimpleVulcanTest.prg

- n **New Vulcan.NET class library**
- n **Add a reference to nunit.framework**
 - à should be in the GAC after installation of NUnit
- n **Write a test class and build**
 - à using the attributes and assertions (static methods) from the framework
 - à now build the assembly
- n **Create a NUnit project in the NUnit GUI**
 - à add the assembly to the project
 - à you can run the test now
- n **Add the NUnit project to your VS solution**
 - à define the NUnit GUI as the default program for NUnit project files

Slide 10 of 27

Installation

Installation of NUnit is pretty simple. Go to www.nunit.org, click on download in the top menu and get the current production release. Choose the Windows installer package. Make sure to pick the version for the .NET framework 2.0 because Vulcan.NET only runs under 2.0.

3. Introduction into NUnit

Test Class

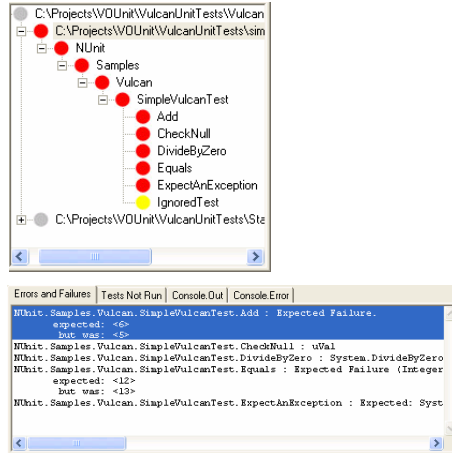
```
Using System
Using NUnit.Framework

[TestFixture];
Class SimpleVulcanTest
    Protected value1 As Int
    Protected value2 As Int

[Setup];
Method Init() As Void Class SimpleVulcanTest
    value1 := 2
    value2 := 3
    Return

[Test];
Public Method Add() As Void Class Simple...
    Local result As Int
    result := value1 + value2
    Assert.AreEqual(6, result,
        "Expected Failure.")
    Return
```

Result in the GUI Runner



Slide 11 of 27

And now?

Make it green! Quick!

Excercise: Work on SimpleVulcanTest.prg method by method and run NUnit each time. Make all tests flash green.

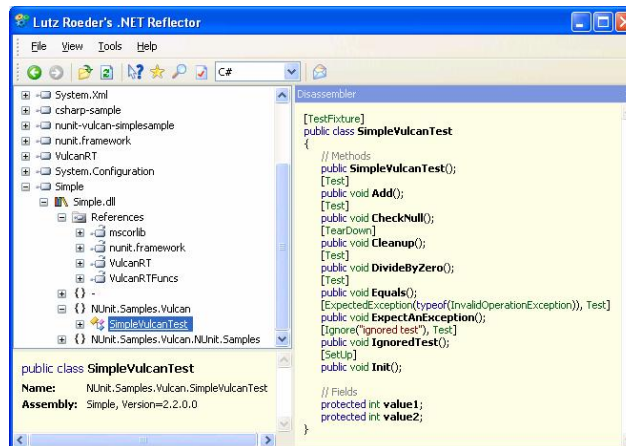
3. Introduction into NUnit

Attributes

- n **TestFixture**
 - à Marks a class as a test case
- n **Setup und TearDown**
 - à These methods are called before and after *each* test method
- n **TestFixtureSetup und TestFixtureTearDown**
 - à These methods are called *once* before and after each test method
- n **Test**
 - à Marks a method a a test
- n **ExpectedException**
 - à You expect an operation to fail
- n More attributes to control the test runs
Ignore, Platform, Category, Explicit

3. Introduction into NUnit

The class in Reflector



Slide 13 of 27

Reflection

Lutz Roeder's Reflector is an outstanding example of what can be achieved with *Reflection* in the .NET framework. The image shows the assembly with our sample test class loaded into the tool. Note the attributes attached to the class itself and the methods. Reflector inspects the code in the assembly to show all this information.

The NUnit runners (GUI and Console) do the same thing. By looking for classes with the *TestFixture* attribute and for methods in these classes with attributes like *SetUp*, *TearDown* or *Test* (all of them defined in the NUnit framework), the runners identify the test cases to execute.

3. Introduction into NUnit

Assertions

The static methods of the Assert class are the most important tools when writing the test cases.

n Equality

à Test for an expected value?
AreEqual and AreNotEqual

n Condition

à Test for the expected state of an object?
IsTrue, IsFalse, IsNull, IsNotNull, IsNaN, IsEmpty, IsNotEmpty

n Identity

à Are two variables pointing to the same CLR object?
AreSame, AreNotSame and Contains

Slide 14 of 27

All methods have numerous overloads.

3. Introduction into NUnit

Assertions

n Type

à Is an object of a particular type?

`IsInstanceOf`, `IsNotInstanceOf`,
`IsAssignableFrom`, `IsNotAssignableFrom`

n Comparison

à Is one object greater/lesser than another?

`Greater` and `Less`

Slide 15 of 27

Enhancements in the upcoming version 2.4 are mainly in the area of assertions:

- `CollectionAssert`
- `FileAssert`
- More Assert methods

4. TDD Sample Session

Task: Test-Driven Development of a *Stack* Class

Expected methods and properties

- n Push()**
 - à Put an object onto the stack
- n Pop()**
 - à Remove an object from the stack
- n Top()**
 - à Return the topmost object but keep it on the stack
- n IsEmpty**
 - à Returns true if there are no objects on the stack

Based on chapter 2 of the book by Jim Newkirk and Alexei Vorontsov.

Slide 16 of 27

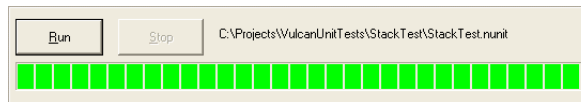
Sounds easy, doesn't it?

Excercise: Write a list of tests you think you should implement.

4. TDD Sample Session

Key Issues of the Sample Session

- n **Separate code and testcode**
 - à Separate assemblies
 - à Test assembly references NUnit.Framework and the actual code
- n **Test-driven Design**
 - à Code follows test, TDD is an agile method, not an unwanted duty!
- n **Principle**
 - à Keep it green



- n **Debugging with unit tests**

Slide 17 of 27

Download a book chapter about this sample

The example is from chapter 2 of the book by Jim Newkirk und Alexei Vorontsov, the authors of NUnit. I only ported the example from C# to Vulcan.NET. You can download this chapter from www.testdriven.com

The Vulcan.NET source of this presentation is included.

4. TDD Sample Session

Debugging

The NUnit GUI Runner can be used to debug code in assemblies.

- n Start the NUnit GUI
- n In Visual Studio: Debug, Attach to Process...
 - à Pick NUnit process
- n Define breakpoints
- n Let the test method run in NUnit

Better alternative: **TestDriven.NET**
(commercial product for professional developers)

5. VOUnit

Developing a Test Framework for Visual Objects has its Problems

Limited language features of VO:

n No attributes

- à How do I mark classes as test fixtures?
- à How do I mark methods as test methods?
- à How do I signal expected exceptions?

n No interfaces

- à see Meinhard Schnoor-Matriciani's session about Design Patterns with VO and Vulcan.NET

n Limited reflection

- à only `ClassList`, `MethodList` etc.

5. VUnit

TestFixture

All tests are in DLLs. The DLLs must contain the following function which identifies the DLL as a test suite to the runner.

```
FUNCTION LoadTestDll() AS STRING STRICT  
RETURN "UnitTest"
```

All test classes must inherit from BaseTest.

```
CLASS FunctionTest INHERIT BaseTest  
  DECLARE METHOD TestDigitsOnly  
  DECLARE METHOD TestDigitsOnly2  
  DECLARE METHOD Setup  
  DECLARE METHOD TearDown
```

The runner regards all classes inheriting from BaseTest as TestFixtures.

5. VOUnit

Attributes

Descriptions of the test methods in the constructor of the test class replace the attributes.

```
METHOD Init() CLASS FunctionTest

    SUPER:Init()

    // Test beschreiben
    cName := "FunctionTest"
    cDesc := "Testing Runtime Functions in AFS Sys"

    aTestMethods: AAdd( TestAttribute(#TestDigitsOnly,2) )
    aTestMethods: AAdd( TestAttribute(#TestDigitsOnly2,1,FALSE,{#Error}) )

    aTestSetUp: AAdd( TestAttribute(#Setup) )
    aTestTearDown: AAdd( TestAttribute(#TearDown) )

RETURN SELF
```

Slide 21 of 27

The parameters to the constructor of TestAttribute are:

- symMethod (Name or the test method)
- nRepeat
- lIgnore
- aExpectedExceptions

5. VOUnit

Assertions

VO does not support static methods. But it has functions.

```
METHOD TestDigitsOnly() AS VOID PASCAL CLASS FunctionTest
    AssertStringEquals( "12345" , DigitsOnly("A1b2d3e4f5","f") )
RETURN

METHOD TestDigitsOnly2() AS VOID PASCAL CLASS FunctionTest
    AssertStringEquals( "1234f5" , DigitsOnly("A1b2d3e4f5","f") )
    _Break( Error() ) // zugelassene Exception s.o. !
RETURN

METHOD Setup() AS VOID PASCAL CLASS FunctionTest
RETURN

METHOD TearDown() AS VOID PASCAL CLASS FunctionTest
RETURN
```

5. VUnit

Console Runner

Expects the test DLLs as command line parameters.



```
C:\WINDOWS\system32\cmd.exe
C:\Projects\UUnitLight\BIN>AF5UnitTest.exe SimpleUnitTest.dll
AF5 UnitTest being loaded ...
FunctionTest # ungueltig # Testing Runtime Functions in AF5 Sys
      SETUP # gueltig # <setup> successfully
      TESTDIGITSONLY # ungueltig # Die Zeichenketten <12345> und <1234f5> sind nicht gleich !
      TESTDIGITSONLY2 # gueltig # caught exception <ERROR>
      TEARDOWN # gueltig # <teardown> successfully
AF5 UnitTest finished.
C:\Projects\UUnitLight\BIN>
```

Status of VUnit

VUnit is an internal beta we already use in our team. What's missing:

- GUI Runner
- More Assert functions

5. VUnit

When do I write new tests for existing projects?

n Bug fixing

à Write a test reproducing the bug first. Then fix the problem.

n Refactoring

à What functionality exactly does the existing code provide?

This is your worst enemy, the biggest, most frustrating and most difficult part of introducing TDD.

n New code

à This is like a new system. The same principles apply. Develop test first.

n Badly documented interface

à How is this component, class or method supposed to be used? Do not write text. Write a test fixture.

Slide 24 of 27

„When we get bug reports, we're trying to follow a methodology of writing a test first that fails. Then we fix the bug and make sure the test passes.“

Don Caton, 08.08.06, auf vo.vops.vulcandotnet

6. More Test Methods

Further areas, tools and methods are

n GUI

- à NUnitForms (tests of System.Windows.Forms applications)
- à Mercury and other commercial tools
- à MVC (Separates visualization and business model)

n Mock objects

- Mock objects mimic the behaviour of external components (e.g. serial interfaces) to enable automated tests
- à NMock or DotNetMock

n Code in databases

- à SQLUnit or NDBUnit

n Web applications

- à NUnitASP

Slide 25 of 27

MVC

Model View Controller. A design pattern that separates GUI code from business logic code.

SQLUnit

Impressive looking open source project. Unfortunately, it is written in Java. I couldn't even get it to run it. And you need to be a Java die-hard to write extensions to it. A similar tool in the .NET world is yet to be written.

A lot of the other tools mentioned on this slide are Nunit add-ons.

7. Integration of TDD into a Software Development Process

TDD is only a first step towards the implementation of an overall quality process (like Extreme Programming) oder another agile method.

In particular:

n Version control systems

à CVSNT, SubVersion, MS Team Services

n Build tools

for continous builds on a central integration system

à NAnt, MSBuild, CruiseControl.NET, FinalBuilder

n Quality assurance

à Code Coverage with NCover and NCoverExplore

à Coding guidelines with FXCop

à Design patterns

Slide 26 of 27

Build Tools

NAnt: The classic open source project in this category. Very comprehensive, a lot of plug-ins available.

MSBuild: Part of the .NET SDK, can load Visual Studio Solution Files (.SLN).

CruiseControl: <http://sourceforge.net/projects/ccnet/>

FinalBuilder: <http://www.finalbuilder.com/>

8. Questions and Discussion



Thank you very much for your time!



FISCHER & CONSULTANTS

Gesellschaft für Softwareentwicklung
und Unternehmensberatung mbH

Martinstrasse 1
44137 Dortmund

www.appfact.com
mifi@appfact.de

Homepage of NUnit: www.nunit.org

Books

Newkirk, Jim; Vorontsov, Alexei: Test-Driven Development in Microsoft .Net, Microsoft Press 2004. (A sample chapter of this book - the one with the stack example - can be downloaded from www.testdriven.com)

Beck, Kent: Extreme Programming.

Fowler, Martin: Refactoring: Improving the Design of Existing Code, Addison Wesley 1999.

Wikipedia

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

http://en.wikipedia.org/wiki/Test-driven_development

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

Articles

Clifton, Mark: Advanced Unit Test, Part V - Unit Test Patterns

<http://www.codeproject.com/gen/design/autp5.asp>

Blogs

Nice blog about books, software and links:

<http://codebetter.com/blogs/darrell.norton/articles/50337.aspx>

Excellent blog on a variety of topics including test-driven development:

<http://weblogs.asp.net/roshero/default.aspx>