eXtreme .NET

Omid Ehsani

Senior Consultant and Trainer

Italian Agile Movement

ehsani@agilemovement.it

Agenda

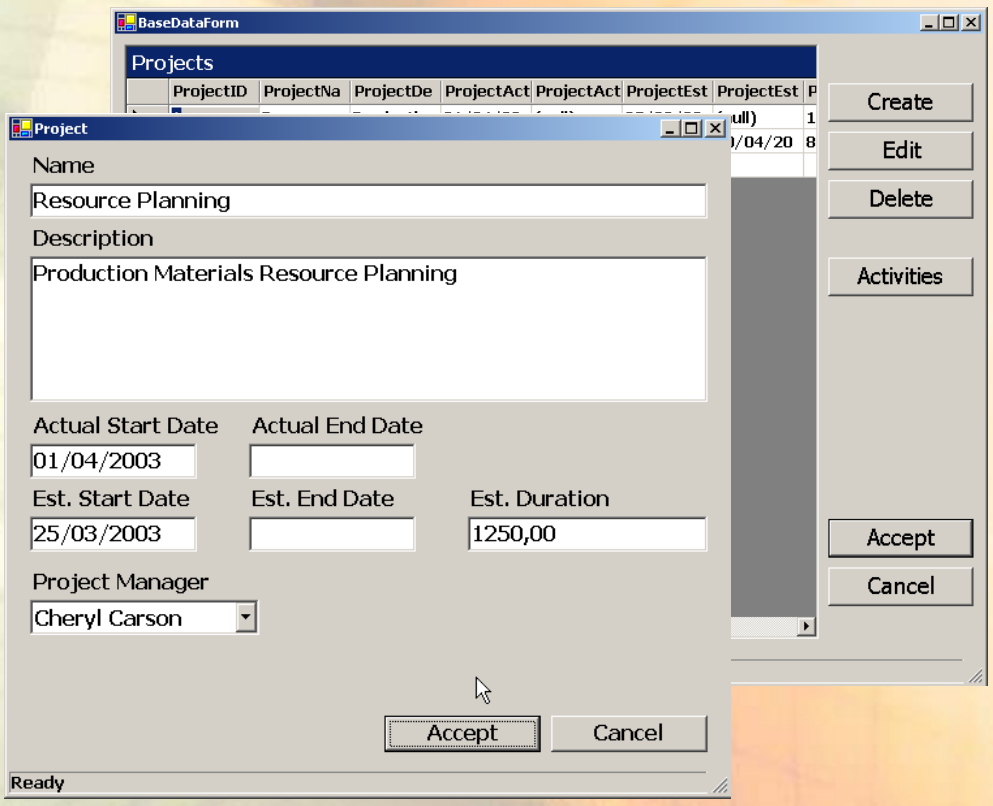
- Case study: OnTime application
- Developing in a team scenario
- Implementing Unit-testing
- NUnit way
- Continuous Integration
- Behind the scenes: Reflection
- Q & A

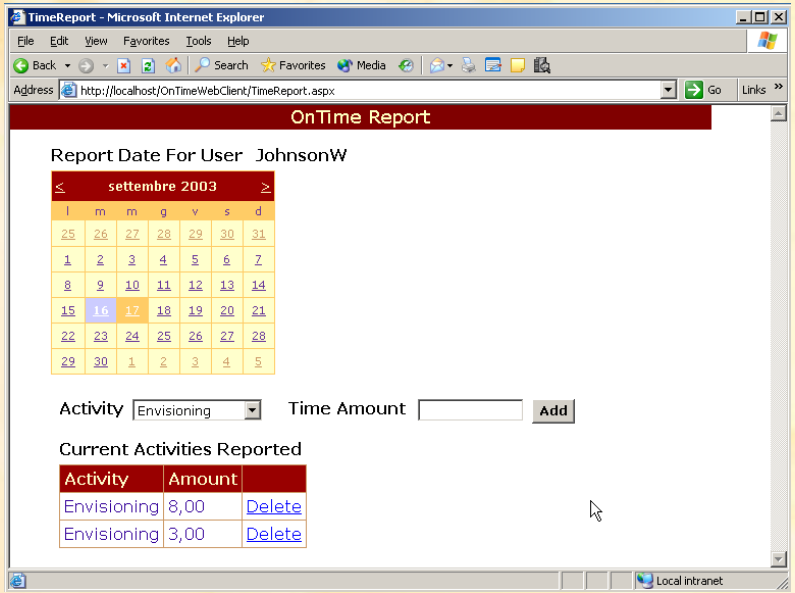
OnTime Application

- Application overview
- Architecture
- Source code insight

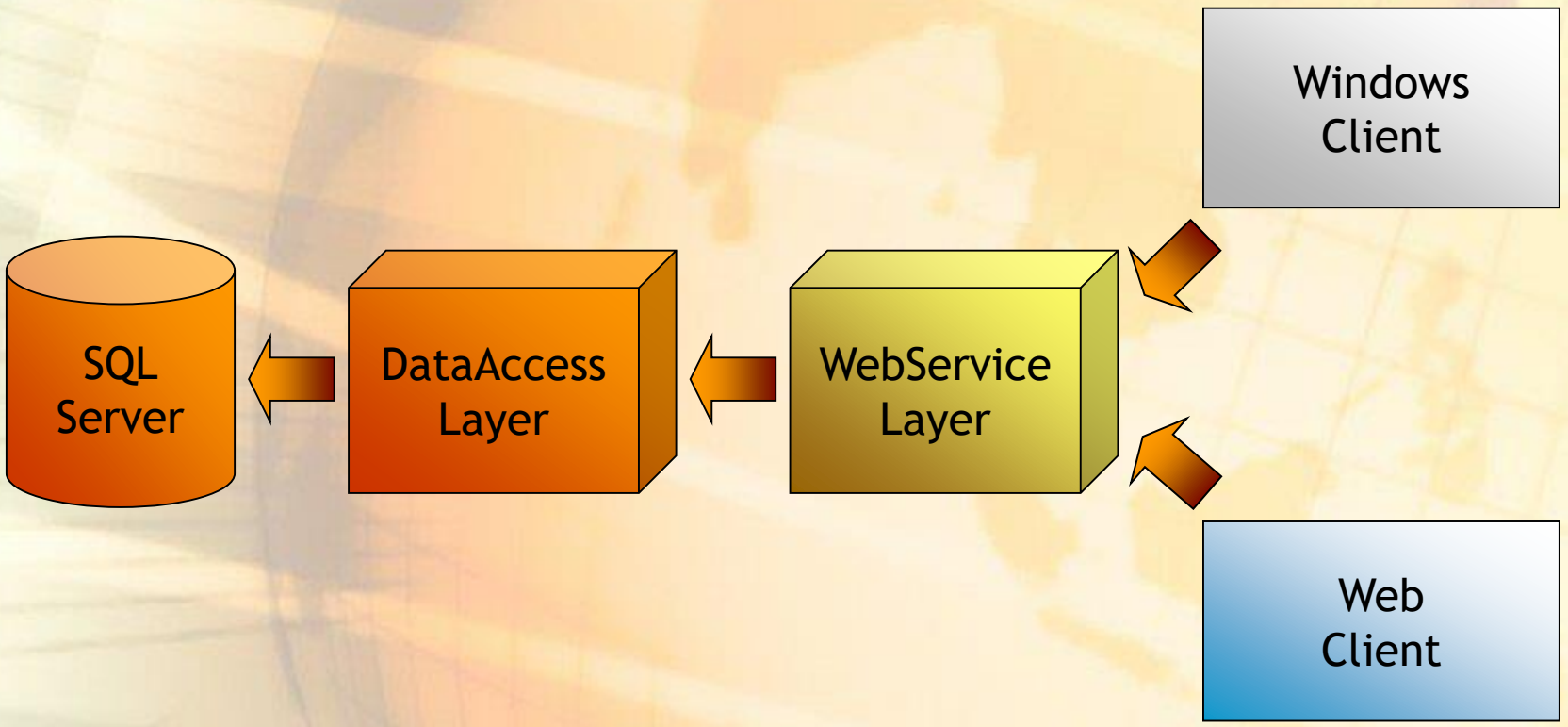
OnTime Overview

- Project time acounting
- Multiple clients





OnTime Architecture



OnTime Code Insight

- Visual Studio 2003 Solution
- Application layers
- DataAccess Application Block
- Typed DataSets
- Windows Forms Inheritance

Developing in Team

- Two distinct teams work toghether
  - Windows Client
  - Web Client
- Data Layer is shared code

Development Scenario

- Web development team is asked to add some new information on the Time Reporting Web Client
- A new developer has joined the team...
- Customer is pressing...
- Time is short...
- Hands on!

Implementing Unit-testing

- Automated test-runner must be able to simulate end user
- Change existing code
  - Separate presentation from process
  - No user interaction in the process components
  - Don’t duplicate code! UI must reuse the process components
- Take a chance to do some refactoring

NUnit Way

- Provides a ready-to-use testing environment
- Little or no impact on existing code
  - Just add some attributes to test classes
- Developer in full control
- Put it to work...

NUnit

A unit-testing framework for all .Net languages

NUnit

- NUnit Overview
- Testing Automation
- Test Attributes
- Assert Class

NUnit Overview

- Porting from JUnit (xUnit)
- Version 2.0 redesigned completely
- Takes advantage of .Net features
  - Custom Attributes
  - Reflection
- Supports .Net Framework 1.0 and 1.1
- Integration with Visual Studio

NUnit Overview

- Forms interface or Console interface
- Test projects
  - XML files
  - Multiple assemblies
  - Multiple configurations
- Reload assemblies on the fly
- XML Output option

Testing Automation

- Use Console test-runner to automate testing operations
- Specify the assembly to run

```
nunit-console /assembly:nunit.tests.dll
```

- Specify the XML output results

```
nunit-console /assembly:nunit.tests.dll
  /xml:console-test.xml
```

- Specify the Transform file

```
nunit-console /assembly:nunit.tests.dll
  /transform:myTransform.xslt
```

Test Attributes

- Identify classes containing test methods
  `[TestFixture]`

```
//C# example
namespace NUnit.Tests
{
  using System;
  using NUnit.Framework;

  [TestFixture]
  public class SuccessTests
  {
    // ...
  }
}
```

Test Attributes

- Identify classes containing test methods

  `<TestFixture()>`

```
'VB example
Imports System
Imports Nunit.Framework

Namespace Nunit.Tests

  <TestFixture()> Public
Class SuccessTests
    ' ...
  End Class
End Namespace
```

Test Attributes

- Identify each test method
  - `[Test]`

```
//C# example
namespace NUnit.Tests
{
  using System;
  using NUnit.Framework;

  [TestFixture]
  public class SuccessTests
  {
    [Test] public void Add()
    { /* ... */ }

    public void TestSubtract()
    { /* backwards compatibility */ }
  }
}
```

Test Attributes

- Identify each test method

  `<Test()>`

```
'VB example
Imports System
Imports Nunit.Framework

Namespace Nunit.Tests

  <TestFixture()> Public Class SuccessTests
    <Test()> Public Sub Add()
    ' ...
    End Sub
  End Class
End Namespace
```

Test Attributes

- Initializing and finalizing test classes

  `[SetUp] / [TearDown]`

```
//C# example
namespace NUnit.Tests
{
  using System;
  using NUnit.Framework;

  [TestFixture]
  public class SuccessTests
  {
    [SetUp] public void Init()
    { /* ... */ }

    [TearDown] public void Dispose()
    { /* ... */ }

    [Test] public void Add()
    { /* ... */ }
  }
}
```

Test Attributes

- Initializing and finalizing test classes
  - `<SetUp()> / <TearDown()>`

```
'VB example
Imports System
Imports Nunit.Framework

Namespace Nunit.Tests

  <TestFixture()> Public Class SuccessTests
    <SetUp()> Public Sub Init()
    ' ...
    End Sub

    <TearDown()> Public Sub Dispose()
    ' ...
    End Sub

    <Test()> Public Sub Add()
    ' ...
    End Sub
  End Class
End Namespace
```

Test Attributes

- Tests throwing exceptions

  `[ExpectedException(...)]`

```
//C# example
namespace NUnit.Tests
{
  using System;
  using NUnit.Framework;

  [TestFixture]
  public class SuccessTests
  {
    [Test]
    [ExpectedException(typeof(InvalidOperationException))]
    public void ExpectAnException()
    { /* ... */ }
  }
}
```

Test Attributes

- Tests throwing exceptions

  `<ExpectedException(...)>`

```
'VB example
Imports System
Imports Nunit.Framework

Namespace Nunit.Tests

  <TestFixture()> Public Class SuccessTests
    <Test(), ExpectedException(GetType(Exception))>
      Public Sub ExpectAnException()
    ' ...
    End Sub
  End Class
End Namespace
```

Test Attributes

- Ignoring tests temporarily

  `[Ignore("...")]`

```
//C# example
namespace NUnit.Tests
{
  using System;
  using NUnit.Framework;

  [TestFixture]
  [Ignore("Ignore a fixture")]
  public class SuccessTests
  {
    // ...
  }
}
```

Test Attributes

- Ignoring tests temporarily

  `<Ignore("...")>`

```
'VB example
Imports System
Imports Nunit.Framework

Namespace Nunit.Tests

  <TestFixture(), Ignore("Ignore a fixture")>
  Public Class SuccessTests
    ' ...
  End Class
End Namespace
```

The Assert Class

- You can use many different conditions to assert the correctness of test results

```
Assert.IsTrue( bool );
Assert.IsFalse( bool );
Assert.IsNull( bool );
Assert.IsNotNull( bool );
Assert.AreSame( object, object );
...
```

The Assert Class

```
...
Assert.AreEqual( object, object );
Assert.AreEqual( int, int );
Assert.AreEqual( double, object );
...
Assert.Fail();
```

And still...

NUnit 2.0 is an excelent example of idiomatic design.

Most folks who port xUnit just translated the Smalltalk or Java version.

That's what we did with NUnit at first, too.

This new version is NUnit as it would have been done had it been done in C# to begin with.

Kent Beck

Continuous Integration

- Basic concepts
- Benefits

Continous Integration Basics

- Keep a single place for all source code
- Automate the build process
- Automate the testing
- Make sure anyone can get a current “good” executable

Continous Integration Benefits

- Identify integration bugs early
  - They can be very hard to catch
  - They could manifest after weeks or months
- Early bug resolution reduces significantly the development efforts
- Make an informed choice:
  - Release the feature at all costs
  - Don’t release the feature (and the bugs)

Behind the scenes

- Reflection is the “Key Technology” of NUnit
- Custom Attributes are:
  - Elegant
  - Not invasive
  - Easily removable
- Let’s have a look at how things go...

Q & A

Additional resources:

http://www.agilemovement.it

http://www.nunit.org

http://nunit.sourceforge.net

http://nant.sourceforge.net