Ambientes de Desenvolvimento Avançados

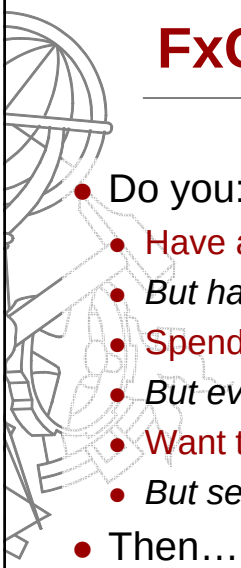
http://www.dei.isep.ipp.pt/~jtavares/ADAV/ADAV.htm

Aula Teórico-Prática

Engenharia Informática

2006/2007

José António Tavares
jrt@isep.ipp.pt



FxCop - Why use FxCop?

- Do you:
  - Have a well defined coding standards
  - *But have no way of enforcing those standards?*
  - Spend much time writing code
  - *But even more time editing code?*
  - Want to have your applications run smoothly
  - *But seem to always be held back by errors?*
- Then…FxCop is for you!

FxCop - What is FxCop?

- Began as an internal Microsoft Solution
- Enforces adherence to .NET Framework Design Guidelines
- Available free http://www.gotdotnet.com/team/fxcop.
- Uses “Introspection”
  - Faster analysis
  - Multi-thread analysis
- Contains over 200 rules
- Ability to create custom rules

Testing - Challenges with Quality Assurance

- Applications need testing
- In the past:
  - Visual Studio focused on software development
  - Light support for testing
  - Required other Microsoft or third-party products
- Improvements address these issues!

Testing - Supported Test Types

- Unit tests
- Web tests
- Generic tests
- Load tests
- Manual tests
- Automated tests and groups of tests can be run from the command line

JUnit - What is it?

- De facto Java unit testing framework
- Integrated nicely with IDEs and Ant
- Easy to learn

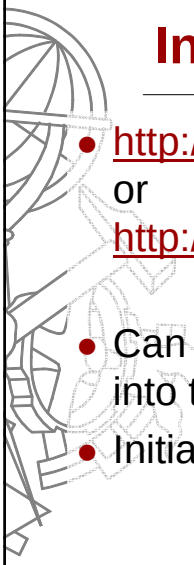
- http://www.junit.org/index.htm

Our first unit test

```
package org.example.antbook.common;

import junit.framework.TestCase;

public class SearchUtilTest extends TestCase {

    public void testSearch() throws Exception {
        // right API?
        Document[] docs =
                  SearchUtil.findDocuments("erik");

        assertTrue(docs.length > 0);
    }
}
```


Lean and green

```
package org.example.antbook.common;

public class SearchUtil {

    public static final Document[]
                    findDocuments(String queryString)
                         throws SearchQueryException,
                                SystemException {
        Document[] results = new Document[1];
        return results;
    }
}
```



Introducing NUnit v2.2 for .NET

- http://sourceforge.net/projects/nunit
or
http://www.nunit.org/

- Can test any .Net Language that is compiled into the CRL (Common Runtime Language)
- Initially ported from JUnit

Introducing NUnit v2.1 for .NET

- NUnit has been completely redesigned to take advantage of many .NET language features
  - custom attributes
  - other reflection related capabilities
- NUnit brings xUnit to all .NET languages

NUnit

- Open source, free for all to use
- Individual TestCase classes and methods are marked using .Net Attributes
- "Suites" of tests and created through reflection; unit-test your entire application with one mouse click.
- Nunit gives you a "TestRunner" GUI Application that runs your tests and presents the results.
- Download at nunit.sourceforge.net

Getting Started with NUnit

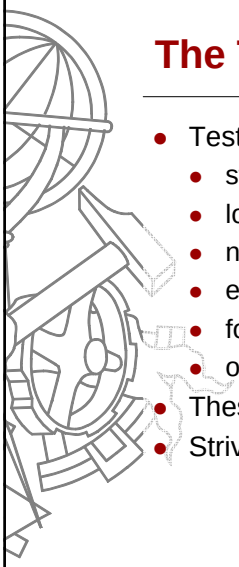
- Install NUnit (nunit.sourceforge.net)
- Add a new class library project to your project's main solution
- Add a Reference to NUnit
- Add a Project Reference to your Unit Test Project that references the assembly you are testing

Test Driven Development

- Test Driven Development says:
  1. Stub out the class
  2. Write a Test that demonstrates one intent
  3. Fail the Test (as expected)
  4. Change the class *just enough to pass the test*
  5. Pass the Test
  6. Return to 2, until all intentions are expressed
- Think of it as Prototyping the actual usage of your code before you write it

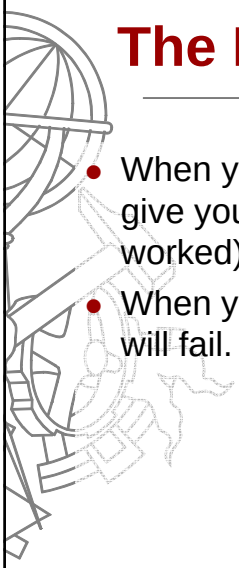
Comparing Test-Later to Test-First

Test-Later	Test-First
• Structure the code "in your head" or using design artifacts • Verify this structure mentally, as/if you think of it and/or have the time. • Automated tests are hard to add (design not intended for them) • Intentions are recorded separately from the code	• Structure is built by coding, and is recorded in the tests. • Every aspect of the structure is verified, using the machine. • Code is designed to be testable, inherently, from the outset. • Tests record and provide examples of intent



The Technological Difference

- Testability is related to:
  - strong cohesion
  - loose coupling
  - no redundancy
  - encapsulation
  - focus
  - objects are responsible for themselves
- These are good programming practices.
- Striving for good tests gives us better code.



The Psychological Difference

- When you write your tests after the fact, all they can give you is bad news (you already thought it worked).
- When you write them before the fact, you know they will fail. Getting the test to run feels good.

Test Driven Development

Laboratório .Net – ISEP

Rui Quintino
rquintino@netcabo.pt

Paulo Oliveira
pjcoliveira@sapo.pt

Test Driven Development

Current status	Stable
Created date	01/26/2004
Audience	Beginner, Intermediate, Advanced
License	View license
Language	C#
Technology	ASP.NET, Web services
Access	Public

`http://workspaces.gotdotnet.com/tdd`

Um ficheiro com o capítulo 2 do livro estará disponível na página da disciplina

Agenda

- O que é o Test Driven Development
- Motivação
- Origem
- Processo
- Demo (NUnit & TDD)
- Vantagens
- Best Practices
- Desafios
- Demo (NUnitForms)
- Questões

Requisitos

- Conhecimento Frameworks Object Oriented

O que é o TDD?

- Técnica de desenvolvimento
- Teste antes do código!
- Testes unitários
  - Testam funcionalidade de módulo/classe em isolamento
- Testes pelo/para Programador
  - Mesma linguagem/Mesmo IDE

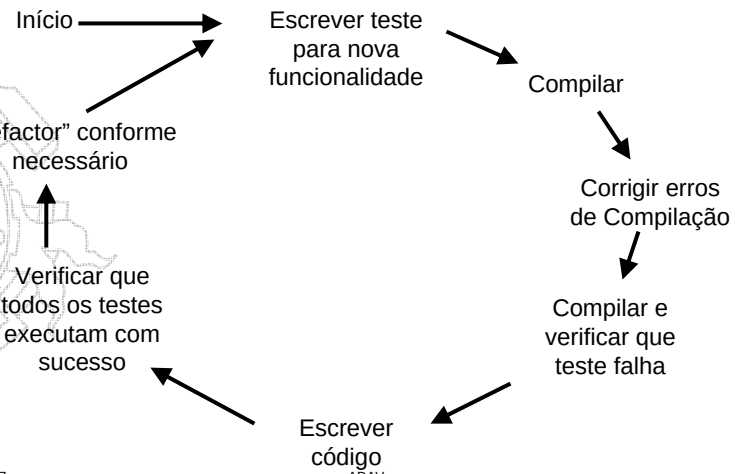
Motivação para o TDD

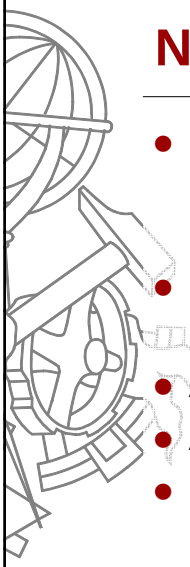
- Melhorar qualidade do código produzido
- Diminuir defeitos
- Diminuir risco
- Potenciar/Encorajar a Mudança!

Origem do TDD

- Uma das práticas principais do eXtreme Programming (XP) – Kent Beck, Martin Fowler – chamadas “Metodologias Ágeis”
- Exploração do potencial de frameworks OO actuais (Java, .Net,…)
- Testes Unitários

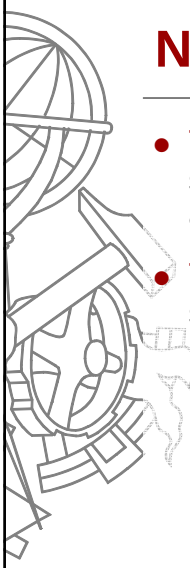
Processo TDD





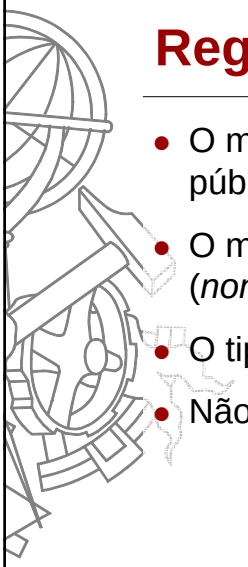
NUnit 2.2

- Introdução
  - O que é
  - GUI e Command Line
- Instalação
  - Testes iniciais
- Assertions
- Attributes
- Demo



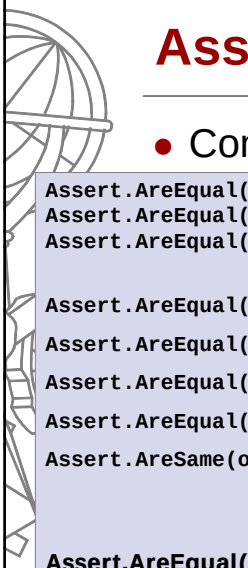
NUnit run-time entities

- Test class
  são as classes que possuem código de teste; cada assembly tem pelo menos uma classe de teste;
- Test methods
  são os casos de teste; cada método de teste representa um simple caso de teste que corre independentemente dos outros casos de teste.



Regras dos métodos de teste

- O método necessita de ser declarado como público;
- O método tem de ser um método de instância (*nonstatic*);
- O tipo de retorno é *void*;
- Não possui quaisquer argumentos.



Assertions

- Comparisons

```
Assert.AreEqual(int expected, int actual );
Assert.AreEqual(int expected, int actual, string message );
Assert.AreEqual(int expected, int actual, string message,
                object[] parms);

Assert.AreEqual(decimal expected, decimal actual );

Assert.AreEqual(float expected, float actual, float tolerance );

Assert.AreEqual(double expected, double actual, double tolerance );

Assert.AreEqual(object expected, object actual );

Assert.AreSame(object expected, object actual );



Assert.AreEqual( 5, 5.0 )
```

Assertions

- Condition Tests

```
Assert.IsTrue( bool condition );
Assert.IsTrue( bool condition, string message );
Assert.IsTrue( bool condition, string message, object[] parms );

Assert.IsFalse( bool condition);
Assert.IsFalse( bool condition, string message );
Assert.IsFalse( bool condition, string message, object[] parms );

Assert.IsNull( object anObject );
Assert.IsNull( object anObject, string message );
Assert.IsNull( object anObject, string message, object[] parms );

Assert.IsNotNull( object anObject );
Assert.IsNotNull( object anObject, string message );
Assert.IsNotNull( object anObject, string message, object[] parms );
```


Assertions

- Utility Tests

```
Assert.Fail();
Assert.Fail( string message );
Assert.Fail( string message, object[] parms );

Assert.Ignore();
Assert.Ignore( string message );
Assert.Ignore( string message, object[] parms );
```

```
public void AssertStringContains( string expected, string actual ) {
 AssertStringContains( expected, text, string.Empty );
}

public void AssertStringContains( string expected, string actual,
                                  string message )
{
  if ( actual.IsIndexOf( expected ) < 0 )
        Assert.Fail( string message );
}
```

Attributes

- TestFixture
- Test
- Setup/Teardown
- Expected Exception
- Category
- Explicit
- Suite
- Ignore

Attributes – Test Fixture

Para ser uma test fixture, as classes devem obedecer a regras.

- Deve ser um Public Type senão o NUnit não o encontra.
- Deve ter um construtor publico senão o NUnit não o consegue criar.
- O construtor não pode ter efeitos secundários porque o NUnit pode ter de criar várias instancias da classes durante uma sessão de testes.

```
namespace NUnit.Tests
{
	using System;
	using NUnit.Framework;

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

Attributes - Test

- Compatibilidade com a versão 1.x
- Assinatura do método

```
public void MethodName()
```

```
namespace NUnit.Tests
{       using System;
        using NUnit.Framework;

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

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

        }
}
```


Attributes – Setup/TearDown

- TestFixtureSetup/TestFixtureTearDown

```
namespace NUnit.Tests
{
        using System;
        using NUnit.Framework;

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

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

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

Attributes – Setup/TearDown

- Setup/TearDown
- Herança – os metodos da classe base são executados

```
namespace NUnit.Tests
{
        using System;
        using NUnit.Framework;
        [TestFixture] public class SuccessTests
        {
                [SetUp] public void Init()
                { /* ... */ }
                [TearDown] public void Dispose()
                { /* ... */ }
                [Test] public void Add()
                { /* ... */ }
        }
}
```


Attributes – Expected Exception

- ExpectedException

```
namespace NUnit.Tests
{
      using System;
      using NUnit.Framework;
      [TestFixture] public class SuccessTests
      {
        [Test] [ExpectedException(typeof(InvalidOperationException))]
        public void ExpectAnException()
        {/* ... */ }
      }
}
```

Attributes - Category

- TestFixture e Test Category

```
Test Fixture Syntax
namespace NUnit.Tests {
        using System;
        using NUnit.Framework;
        [TestFixture] [Category("LongRunning")]
        public class LongRunningTests
        { // ... }
}
Test Syntax
namespace NUnit.Tests {
        using System;
        using NUnit.Framework;
        [TestFixture] public class SuccessTests {
                [Test] [Category("Long")]
                public void VeryLongTest()
                { /* ... */ }
        }
}
```


Attributes - explicit

```
Test Fixture Syntax
namespace NUnit.Tests
{
        using System;
        using NUnit.Framework;
        [TestFixture, Explicit] public class ExplicitTests
        { // ... }
}
Test Syntax
namespace NUnit.Tests
{
        using System;
        using NUnit.Framework;
        [TestFixture] public class SuccessTests
        {
                [Test, Explicit] public void ExplicitTest()
                { /* ... */ }
        }
}
```

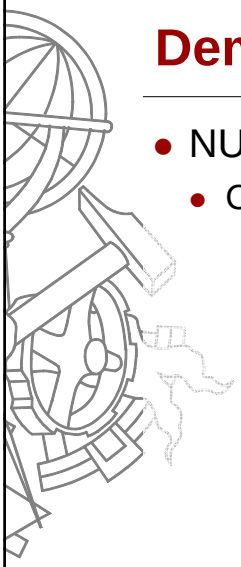
Attributes - Suite

- Compatibilidade
- Não há GUI

```
namespace NUnit.Tests
{
        using System;
        using NUnit.Framework;
        public class AllTests
        {
                [Suite] public static TestSuite Suite
                {
                    get {
                         TestSuite suite = new TestSuite("All Tests");
                         suite.Add(new OneTestCase());
                         suite.Add(new Assemblies.AssemblyTests());
                         suite.Add(new AssertionTest());
                         return suite; }
                }
        }
}
```

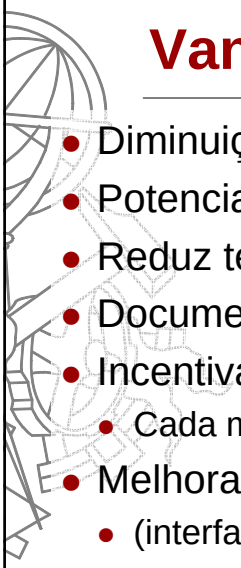

Attributes - Ignore

```
Test Fixture Syntax
namespace NUnit.Tests {
        using System;
        using NUnit.Framework;
        [TestFixture] [Ignore("Ignore a fixture")]
        public class SuccessTests
        { // ... }
}
Test Syntax
namespace NUnit.Tests {
        using System;
        using NUnit.Framework;
        [TestFixture] public class SuccessTests
        {
                [Test] [Ignore("Ignore a test")]
                public void IgnoredTest()
                { /* ... */ }
        }
}
```



Demo!

- NUnit 2.2 & TDD
  - Calculadora



Vantagens

- Diminuição de erros e defeitos
- Potencia Refactoring agressivo
- Reduz tempos de Debugging
- Documentação “Executável”
- Incentiva à Reusabilidade
  - Cada módulo tem pelo menos 2 clientes
- Melhora skills de arquitectura e desenho
  - (interface based design, padrões)

Vantagens (cont.)

- Separação clara de competências entre módulos e classes
- Programação por “intenção”
- Medição efectiva de progresso do projecto
- Maior confiança na evolução do produto
- Adesão cada vez maior (mundo Open Source)
- Vantagem competitiva para a empresa e para o programador!

Vantagens (cont.)

- Ciclos desenvolvimento menores/esforço mais nivelado
- Bugs ficam claramente registados – testes!
- Adaptação a ambientes de integração contínua
- Conceito multi-plataforma

Best Practices

- Teste em 1º Lugar!
- Cobertura de código tão extensa quanto possível
- Testar tudo o que possa falhar
- Teste de classes em isolamento
  - Testes imunes a influências externas
- Aplicação de Model-View-Controller para teste de Interfaces

Desafios

- Testes de BD
- Testes de Interfaces Gráficos
- Testes de Aplicações Web
- Testes em Isolamento
  - Mock frameworks
- Grandes Aplicações

NUnit Forms 1.3.1

- Criar testes unitários

```
Form form = new Form();
form.Show();
```

```
ButtonTester button = new ButtonTester("buttonName");
ButtonTester button = new ButtonTester("buttonName", "formName");
```

```
ControlTester textBox = new ControlTester("nameOfSomeTextBox");
Assertion.AssertEquals("defaultText", textBox["Text"];
textBox["text"] = "newText";
```

```
ControlTester button = new ControlTester("nameOfSomeButton");
button.FireEvent("Click");
```


Control Testers

- Extender a classe ControlTest
- Implementar 3 construtores
- Implementar uma propriedade Properties para suporte do Recorder
- Implementar propriedades explicitas
- Criar NUnit tests

Control Testers

```
public class ComboBoxTester : ControlTester
public ComboBoxTester(string name, Form form) : base(name, form){}
public ComboBoxTester(string name, string formName) : base(name, formName){}
public ComboBoxTester(string name) : base(name){}

public ComboBox Properties {
  get {
    return (ComboBox) Control;
  }
}

public void Select(int i) {
   Properties.SelectedIndex = i;
}

public string Text {
  get {
    return Properties.Text;
  }
}
```


Demo!

- NUnit Forms 1.3.1
  - Calculadora GUI

Ferramentas

- Frameworks xUnit (NUnit, MBUnit, csUnit, NUnitASP)
- Test Driven .Net (plugin Visual Studio)
- Visual Studio 2005 Team System
- Cruise Control
- NAnt
- …

Referências

- Kent Beck, "Test Driven by Example“
- Martin Fowler, "Refactoring: Improving the design of existing code"
- James W. Newkirk & Alexei A. Vorontsov,"Test-Driven Development in Microsoft.Net"
- Ronald E. Jeffries, "Extreme Programming Adventures in C#"

Links

James Newkirk
blogs.msdn.com/jamesnewkirk
Brian Marick
www.testing.com/cgi-bin/blog
Jonathan de Halleux
http://blog.dotnetwiki.org/
Jose Almeida
blogs.msdn.com/josealmeida/
Scott Densmore –
blogs.msdn.com/scottdensmore
Code Project
www.codeproject.com/dotnet/tdd_in_dotnet.asp
TestDriven.com
www.testdriven.com

Questões

?