Ferramentas para Testes Automatizados em Java e C++

Fabiana Piesigilli

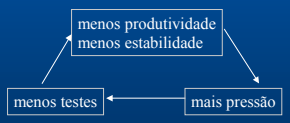
Baseado na apresentação
"Qualidade e Produtividade em Projetos Java"
de Helder da Rocha
no Brasil@JavaOne 2002

Para que testar?

- **Qualidade: Código testado é mais confiável**
  - **Como saber se o recurso funciona sem testar?**
- **Coragem para mudar: De que adianta a OO isolar a interface da implementação se o programador tem medo de mudar a implementação?**
  - **Como saber se ainda funciona após refatoramento?**
- **Saber quando o projeto está pronto. Testes são requisitos "executáveis". Escreva-os antes. Quando todos rodarem 100%, o projeto está concluído!**

Problema com testes (e a solução)

- Todos sabem: devem ser escritos;
- Poucos o fazem, e por quê não ?
  - Estou com muita pressa
- Mas isto cria um círculo vicioso



- Como quebrar este ciclo: criando um ambiente simples de testes. Depois de fazer os primeiros testes, o hábito vem para ficar.

O que são ferramentas para testes automáticos?

- **São programas que avaliam se outro programa funciona como esperado e retornam resposta do tipo "sim" ou "não".**

  Ex: um main() que cria um objeto de uma classe testada, chama seus métodos e avalia os resultados

- **Validam os requisitos de um sistema**

JUnit: O que é?

- **JUnit é um "framework" que facilita o desenvolvimento e execução de testes de unidade em código Java**
  - **Uma API para construir os testes**
    - **Classes Test, TestCase, TestSuite, etc. oferecem a infraestrutura necessária para criar os testes.**
    - **Métodos assertTrue( ), assertEquals( ), fail( ), etc. são usados para testar os resultados.**
  - **Aplicações para executar testes (os TestRunners)**
    - **Rodam testes individuais e suites (conjuntos de testes). Versões texto, Swing e AWT.**

JUnit: Para que serve?

- **"Padrão" para testes de unidade em Java**
- **Desenvolvido por Kent Beck (o guru do XP) e Erich Gamma (do GoF "Design Patterns")**
- **Facilita:**
  - **A criação e execução automática de testes**
  - **A apresentação dos resultados**
- **JUnit pode verificar se cada método de uma classe funciona da forma esperada**
  - **Permite agrupar e rodar vários testes ao mesmo tempo**
  - **Na falha, mostra a causa em cada teste**
- **Serve de base para extensões**

CppUnit

- **CppUnit é a versão para C++ do JUnit (mas não é idêntico)**

  http://cppunit.sourceforge.net

- **A saída dos testes pode ser em XML ou em formato de texto para testes automáticos, e com interface visual para testes supervisionados**

7

JUnit: Como usar

- **Crie uma classe que estenda junit.framework.TestCase**

  class SuaClasseTest extends junit.framework.TestCase {...}
- **Para cada método xxx(args) a ser testado defina um método public void testXxx() na classe de teste**

  SuaClasse:

  public boolean equals(Object o) { ... }

  SuaClasseTest:

  public void testEquals() {...}
- **Sobrescreva o método setUp(), se necessário inicialização comum a todos os métodos.**
- **Sobrescreva o método tearDown(), se necessário para liberar recursos como streams, apagar arquivos, etc.**

8

JUnit: Como usar

- **Use testXxx() para testar seu método xxx().**
- **Utilize os métodos de TestCase**

  assertEquals(objetoEsperado, objetoRecebido),

  assertTrue(valorBooleano),

  assertNotNull(objeto)

  assertSame(objetoUm, objetoDois),

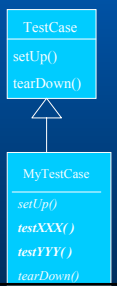
  fail (), ...
- **Exemplo:**

```
public class CoisaTest extends TestCase {
        // construtor padrão omitido
        private Coisa coisa;
        public void setUp() {
                coisa = new Coisa(" Bit");
        }
        public void testToString() {
                assertEquals("< coisa> Bit</ coisa>", coisa.toString());
        }
}
```

9

JUnit: Como funciona

- **O TestRunner recebe uma subclasse de junit.framework.TestCase**
- **Usa reflection para descobrir seus métodos**
- **Para cada método testXXX(), executa:**
  1. **o método setUp()**
  2. **o próprio método testXXX()**
  3. **o método tearDown()**
- **O test case é instanciado para executar um método testXXX() de cada vez.**
- **As alterações que ele fizer ao estado do objeto não afetarão os demais testes**
- **Método pode** *terminar*, *falhar* **ou provocar** *exceção*



JUnit: Exemplo

```
package junitdemo;
import java. io.*;
public class TextUtils {
    public static String removeWhiteSpaces( String text)
            throws IOException {
                StringReader reader = new StringReader( text);
                StringBuffer buffer = new StringBuffer( text. length());
                int c;
                while( (c = reader. read()) != -1) {
                        if (c == ' '|| c == '\ n'|| c == '\ r'|| c == '\ f'|| c == '\ t') {
                                /* não faz nada */
                        }
                        else {
                                buffer. append(( char) c);
                        }
                }
                return buffer.toString();
    }
}
```

11

JUnit: Exemplo

Um test case para a classe

```
package junitdemo;
import junit. framework.*;
import java. io. IOException;
public class TextUtilsTest extends TestCase {
    public TextUtilsTest( String name) {
            super( name);
    }
    public void testRemoveWhiteSpaces()
      throws IOException {
            String testString = "one, ( two | three+) , "+"(((four+ |\ t five)?\ n \n, six?";
            String expectedString = "one,( two| three+)"+",(((four+|five)?, six?";
            String results = TextUtils. removeWhiteSpaces( testString);
            assertEquals( expectedString, results);
    }
}
```

12

Portando o test case para o CppUnit

```
#include "TestUtils.h"
#include <cppunit/TestCase.h>
#include <cppunit/TestAssert.h>
using namespace std;
using namespace cppDemo;
class TextUtilsTest: public CppUnit::TestCase {
 public:
   TextUtilsTest ( std::string name ) : CppUnit::TestCase( name ) {}
   void testRemoveWhiteSpaces () {
     string testString ("one, ( two | three+), (((four+ |\ t five)?\ n \n, six? ");
     string expectedString ("one,( two| three+),(((four+|,five)?, six? ");
     string results = TextUtils.removeWhiteSpaces( testString);
     CPPUNIT_ASSERT_EQUAL (expectedString, results);
   }
}
```


fail: Teste situações de falha

- **É tão importante testar o cenário de falha do seu codigo quanto o sucesso**
  - **Método fail() provoca uma falha**
  - **Use para verificar se exceções ocorrem quando se espera que elas ocorram**
- **Exemplo**

```
public void testEntityNotFoundException() {
        resetEntityTable(); // não há entidades para resolver!
        try {
                // A chamada do método a seguir deve provocar uma exceção!
                ParameterEntityTag tag = parser.resolveEntity(" bogus");
                fail(" Deveria ter causado EntityNotFoundException!");
        }
        catch (EntityNotFoundException e) {
                // sucesso: a exceção ocorreu conforme esperado
        }
}
```

- **Falhas são testadas no CppUnit com a macro CPPUNIT_FAIL (mensagem)**

JUnit: Fixtures

- **São os dados reutilizados por vários testes**
  - **Inicializados no setUp() e destruídos no tearDown() (se necessário)**

```
public class AttributeEnumerationTest extends TestCase {
        String testString;
        String[] testArray;
        AttributeEnumeration testEnum;
        public void setUp() {
                testString = "( alpha| beta| gamma)";
                testArray = new String[]{" alpha", "beta", "gamma"};
                testEnum = new AttributeEnumeration( testArray);
        }
        public void testGetNames() {
                assertEquals( testEnum. getNames(), testArray);
        }
        public void testToString() {
                assertEquals( testEnum. toString(), testString);
        }
        (...)
```


- **Extensão** *JXUnit* **(jxunit.sourceforge.net) permite manter dados de teste em arquivo XML (*.jxu) separado do código => Mais flexibilidade. Permite escrever testes mais rigorosos, com muitos dados**

CppUnit: Fixtures

- **Para usar fixtures, sua classe precisa herdar não de TestCase, mas de TestFixture**
- **Os métodos setUp e tearDown funcionam da mesma maneira, mas devem ser protected.**
- **Exemplo:**

```
class TextUtilsTest : public CppUnit:: TestFixture {
        (...)
        protected:
          void setUp() { ...}
          void tearDown() { ... }
 }
```


JUnit: A Classe TestSuite

- **Permite executar uma coleção de testes: Método addTest(Test) adiciona um teste na lista**
- **Padrão de codificação (usando reflection):**
  - **retornar um TestSuite em cada test-case**

```
public static Test suite() {
        return new TestSuite( SuaClasseTest.class);
}
```

  - **criar uma classe AllTests que combina as suites:**

```
public class AllTests {
        public static Test suite() {
                TestSuite testSuite = new TestSuite(" Roda tudo");
                testSuite.addTest( pacote.AllTests.suite());
                testSuite.addTest( MinhaClasseTest.suite());
                testSuite.addTest( SuaClasseTest.suite());
                return testSuite;
        }
}
```


CppUnit: TestSuite

- **Para criar uma suite com 2 ou mais teste:**

```
CppUnit::TestSuite suite;
suite.addTest( new CppUnit::TestCaller<NomeDaSuaClasseDeTeste>(
        "oNomeDoSeuTeste",  &SuaClasseDeTeste::seuMetodoDeTeste));
suite.addTest ( OutraClasseDeTeste::suite() );
```

- **O método suite, como no JUnit, também precisa ser público, estático mas deve retornar um ponteiro para um objeto que implementa a interface Test:**

```
public:
 static CppUnit::Test *suite() {
        CppUnit::TestSuite *suiteOfTests = new
                CppUnit::TestSuite("NomeDoSeuTeste" );
        suiteOfTests->addTest (...)
        return suiteOfTests;
 }
```

JUnit: como executar

•Use a interface de texto

```
java -cp junit.jar junit.textui.TestRunner
junitdemo.TextUtilsTest
```

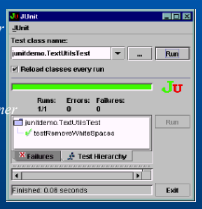
•Ou use a interface gráfica

```
java -cp junit.jar junit.swingui.TestRunner
junitdemo.TextUtilsTest
```

•Use Ant <junit>

•Ou forneça um main():

```
public static void main (String[] args) {
        TestSuite suite = new TestSuite(TextUtilsTest.class);
        junit.textui.TestRunner.run( suite);
}
```



CppUnit: como executar

- **Usando a versão texto:**

```
#include <cppunit/ui/text/TestRunner.h>
#include "SeuTeste.h"
int main( int argc, char **argv) {
    CppUnit::TextUi::TestRunner runner;
    runner.addTest( ExampleTestCase::suite() );
    runner.addTest( ComplexNumberTest::suite() );
    runner.run();
    return 0;
}
```

- **Também há duas versões gráficas: MFC e QT. Consulte a documentação para mais informações.**

Afirmações do J2SDK 1.4 (assertions)

- **São expressões booleanas que o programador define para afirmar uma condição que ele acredita ser verdade**
  - **Nova palavra-chave:** assert
  - **Exemplo:** assert i == 2 : "Erro: i != 2";
  - Afirmações são usadas para validar código (ter a certeza que um vetor tem determinado tamanho, ter a certeza que o programa não passou por determinado lugar)
  - Devem ser usadas durante o desenvolvimento e desligadas na produção (afetam a performance)
  - Não devem ser usadas como parte da lógica do código
  - Provocam um AssertionError quando falham
- **Afirmações são um recurso novo do J2SDK1.4.0**
  - É preciso compilar usando a opção -source 1.4:
    > javac -source 1.4 Classe.java
  - Para executar, é preciso habilitar afirmações (enable assertions):
    > java -ea Classe

JUnit vs. afirmações

- **Afirmações do J2SDK 1.4 são usadas dentro do código**
  - Podem incluir testes dentro da lógica procedural de um programa

```
if (i% 3 == 0) {
        doThis();
} else if (i% 3 == 1) {
        doThat();
} else {
        assert i% 3 == 2 : "Erro interno!";
}
```

  - O AssertionError provocado pode ser encapsulado pelas exceções do JUnit
- **Afirmações do JUnit são usadas em classe separada (TestCase)**
  - Não têm acesso ao interior dos métodos (verificam se a interface dos métodos funciona como esperado)
- **Afirmações do J2SDK1.4 e JUnit são complementares**
  - JUnit testa a interface dos métodos
  - assert testa trechos de lógica dentro dos métodos

Limitações do JUnit

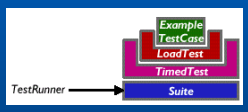
- **Acesso aos dados de métodos sob teste**
  - **Métodos privados e variáveis locais não podem ser testadas com JUnit.**
  - **Dados devem ser pelo menos package-private (friendly)**
- **Soluções com refatoramento**
  - **Isolar em métodos privados apenas código inquebrável**
  - **Transformar métodos privados em package-private**
    - **Desvantagem: quebra ou redução do encapsulamento**
    - **Classes de teste devem estar no mesmo pacote que as classes testadas para ter acesso**
- **Solução usando extensão do JUnit**
  - JUnitX: **usa reflection para ter acesso a dados privados**
    http://www.extreme-java.de/junitx/index.html

JUnitPerf: Testes de Performance

- **Coleção de decorators para medir performance e escalabilidade em testes JUnit existentes** (http://www.clarkware.com)
- **TimedTest**
  - **Executa um teste e mede o tempo transcorrido**
  - **Define um tempo máximo para a execução. O teste falha se a execução durar mais que o tempo estabelecido**
- **LoadTest**
  - **Executa um teste com um número simulado de iterações e usuários concorrentes**
  - **Utiliza timers para distribuir as cargas usando distribuições aleatórias ou intervalos de tempo pré-determinados**
  - **Combinado com TimerTest para medir tempo com carga**
- **ThreadedTest**
  - **Executa o teste em um thread separado**

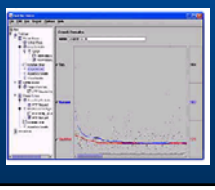
JUnitPerf: Testes de Performance

```
import com.clarkware.junitperf.*;
import junit.framework.*;
public class ExampleLoadTest extends TestCase {
    public ExampleLoadTest( String name) {
        super( name);
    }
    public static Test suite() {
        Timer timer = new ConstantTimer( 1000);
        int users = 10;
        int iters = 10;
        long maxElapsedTime = 20000;
        Test test = new ExampleTestCase("testOneSecondResp");
        Test loadTest = new LoadTest( test, users, iters, timer);
        Test timedTest = new TimedTest( loadTest, maxElapsedTime);
        TestSuite suite = new TestSuite();
        suite. addTest( timedTest);
        return suite;
    }
    public static void main( String args[]) {
        junit. textui. TestRunner. run(ExampleLoadTest.suite());
    }
```



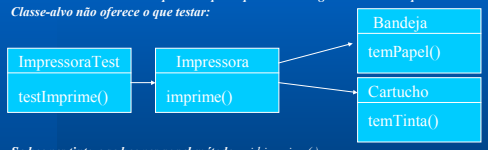
JMeter: Testes de Stress

- *Uma ótima ferramenta open-source para testar os limites de uma aplicação*
  - *Simula carga pesada em servidor, rede ou objeto*
  - *Testa performance sob diferentes tipos de carga em aplicações Web, servidores de FTP, objetos Java, bancos de dados, servlets, scripts em Perl, sistemas de arquivos, e outras aplicações (Java ou não)*
  - *Gera gráficos com resultados para análise*
- *JMeter pode ser usado para simular carga que cause falha em testes decorados com JUnitPerf*
- *http://jakarta.apache.org/jmeter*



Dependência de código-fonte

- *Problema: Como testar componente que depende do código de outros componentes?*
- *Classe-alvo não oferece o que testar:*

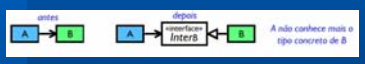


- *Se houver tinta e se houver papel método void imprime( ) deve funcionar*
- *Como saber se há ou não tinta e papel?*

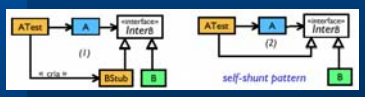
```
public void testImprime( ) {
    Impressora imp = new Impressora();
    imp.imprime( ); // void!
    assert???(???);
}
```


Stubs: objetos "impostores"

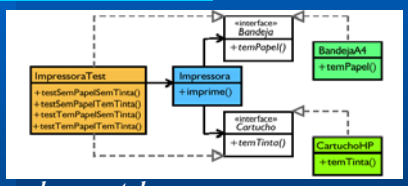
- *É possível remover dependências de código-fonte refatorando o código para usar interfaces*



- *Agora B pode ser substituída por um stub*
  - *BStub está sob controle total de ATest (1)*
  - *Em alguns casos, ATest pode implementar InterB (2)*



Dependência: solução usando stubs



- *Quando usar stubs*
  - *Dependência ainda não está pronta*
  - *Dependências têm estado mutante, imprevisível ou estão indisponíveis durante o desenvolvimento*
    - *BDs, servidores de aplicação, servidores Web, hardware*

Dependências de servidores

- *Usar stubs para simular serviços*
  - *É preciso implementar classes que devolvam as respostas esperadas para diversas situações*
  - *Complexidade pode não compensar investir no desenvolvimento*
  - *Use soluções open-source prontas!*
    - *DBUnit: extensão do JUnit para testar aplicações JDBC http://dbunit.sourceforge.net*
    - *JUnitEE: extensão do JUnit para testar aplicações J2EE http://junitee.sourceforge.net*
- *Usar proxies para serviços reais*
  - *Testa a integração real do componente com seu ambiente*
  - *Solução open-source:*
    - *Cactus: testa integração da aplicação com servlet containers*

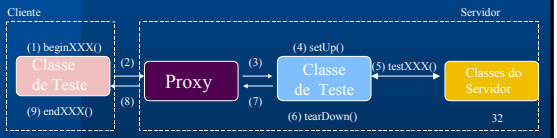
Cactus: Testes de aplicações J2EE

- **Cactus é um framework open-source do projeto Jakarta para testar aplicações que utilizam componentes J2EE**
  - **Componentes Web (Camada de Controle)**
  - **Camada EJB (Model) e cliente (View): indiretamente**

- Cactus estende o JUnit
  - Execução dos testes é realizada de forma idêntica
  - TestCases são construídos herdando de uma **subclasse** de junit.framework.TestCase (Exemplo: ServletTestCase)

Cactus: Testes de aplicações J2EE

- **Cactus testa a integração dos componentes Web com seus containers**
  - **Usa o próprio container como servidor e JUnit como cliente**
    - **Uma cópia do Test Case é instanciada pelo container**
    - **Outra cópia é instanciada pelo JUnit**
  - **Comunicação é intermediada por um proxy (ServletRedirector, JSPRedirector ou FilterRedirector)**
    - **JUnit envia requisições via HTTP para proxy**
    - **Proxy devolve resultado via HTTP e JUnit os mostra**



Testes de interface em servidores Web

- **Testar o funcionamento da aplicação do ponto de vista do usuário**
  - **Verificar se uma página HTML ou XML contém determinado texto ou determinado elemento**
  - **Verificar se resposta está de acordo com dados passados na requisição: testes funcionais do tipo "caixa- preta"**
- **Soluções (extensões do JUnit):**
  - HttpUnit e ServletUnit**:**
    - **permitem testar dados de árvore DOM HTML gerada**
  - JXWeb **(combinação do JXUnit com HttpUnit)**
    - **permite especificar os dados de teste em arquivos XML**
    - **arquivos de teste Java são gerados a partir do XML**
  - XMLUnit: **extensão simples para testar árvores XML**

http://(httpunit| jxunit| xmlunit).sourceforge.net

Concluindo

- **Principais ferramentas comerciais incluem o JUnit: Together, Control Center, Forte, Websphere, etc.**
- **Testes proporcionam:**
  - **Planejamento**
  - **Refinamento do design**
  - **Ritmo saudável**
  - **Design simples**
  - **Lançamentos pequenos**
- **Vale a pena investir tempo para desenvolver e aperfeiçoar a prática constante de escrever testes automatizados**
  - **mais produtividade, maior integração de equipes**
  - **produtos de melhor qualidade, com prazo previsível**
  - **menos stress, mais organização**

Saiba mais

[1] Slides de Helder da Rocha: Implementando eXtreme Programming em Java (http://www.argonavis.com.br).

[2] Slides do Professor Alfredo Goldman (http://grenoble.ime.usp.br/~gold/temp/xp)

[3] **Richard Hightower e Nicholas Lesiecki. Java Tools for eXtreme Programming. Wiley, 2002. Explora as ferramentas Ant, JUnit, Cactus, JUnitPerf, JMeter e HttpUnit usando estudo de caso com processo XP.**

[4] **Jeffries, Anderson, Hendrickson. eXtreme Programming Installed, Addison-Wesley, 2001. Contém exemplos de estratégias para testes.**

[5] **Apache Cactus User's Manual. Contém tutorial para instalação passo-a-passo .**

[6] **Kent Beck, Erich Gamma. JUnit Test Infected: programmers love writing tests. (JUnit docs). Aprenda a usar JUnit em uma hora.**

[7] **Andy Schneider. JUnit Best Practices. JavaWorld, Dec. 2000. O que não fazer para construir bons testes.**

Saiba mais

[8] **Robert Koss, Testing Things that are Hard to Test. Object Mentor, 2002 http://www.objectmentor.com/resources/articles/TestingThingsThatAreHa~9740.ppt. Mostra estratégias para testar GUIs e código com dependências usando stubs.**

[9] **Mackinnon, Freeman, Craig. Endo- testing with Mock Objects. http://mockobjectssourceforge.net/misc/mockobjects. pdf. O autor apresenta técnicas para testes usando uma variação da técnica de stubs chamada de "mock objects".**

[10] **William Wake. Test/ Code Cycle in XP. Part 1: Model, Part II: GUI. http://users.vnet.net/wwake/xp/xp0001/. Ótimo tutorial em duas partes sobre a metodologia "test-first" mostrando estratégias para testar GUIs na segunda parte.**

[11] **Steve Freeman, Developing JDBC Applications Test First. 2001. Tutorial sobre metodolodia test- first com mock objects para JDBC.**

Saiba mais

[12] Martin Fowler. Refactoring: improving the design of existing code. Addison-Wesley, 2000. Cap 4 (building tests) é um tutorial usando JUnit.

[13] Apache JMeter User's Manual. Fonte dos exemplos

[14] Mike Clark, JUnitPerf Docs. Fonte dos exemplos.

[15] CppUnit Cookbook (http://sourceforge.net/projects/cppunit)

37