

Implementando **eXtreme Programming** em **Java**

*Integração contínua e
testes de unidade*



Helder da Rocha
www.argonavis.com.br



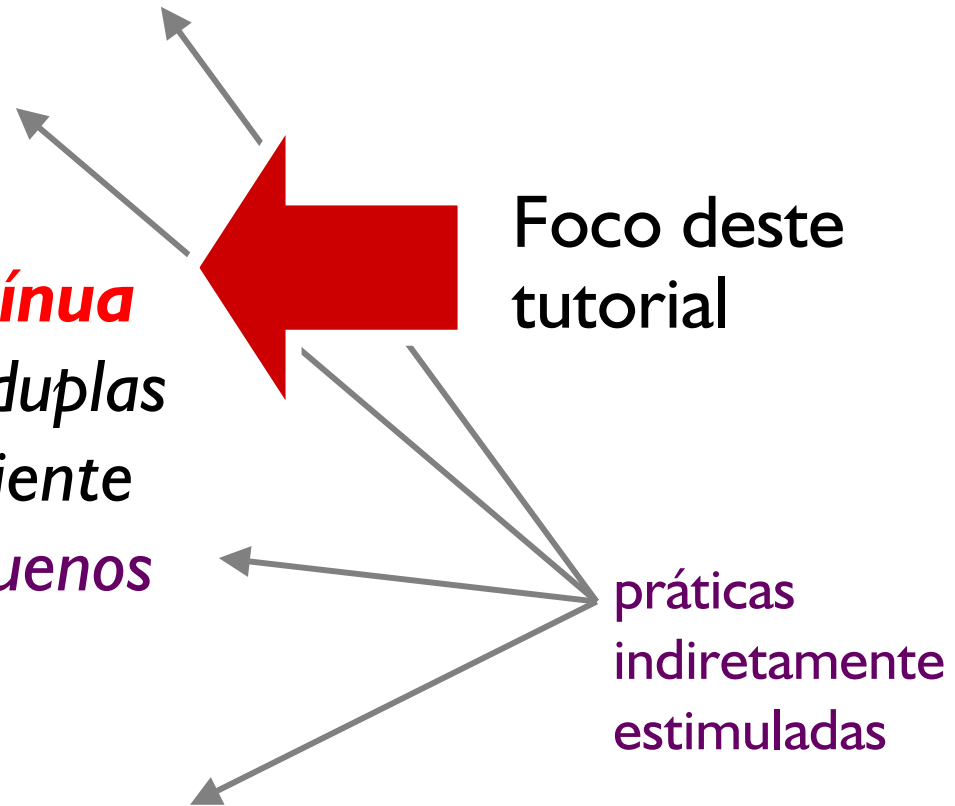
- Apresentar e demonstrar ferramentas **open source** que ajudam a implementar práticas recomendadas pela metodologia **eXtreme Programming (XP)** em projetos Java
 - Integração contínua
 - Testes de unidade
 - Testes de integração
- Apesar do título deste tutorial, as ferramentas (e técnicas) apresentadas não são exclusividade de projetos XP
 - As práticas estimuladas pelas ferramentas e técnicas ajudam a melhorar a **qualidade** de qualquer projeto Java

- *Este é um tutorial técnico destinado a desenvolvedores Java*
 - *Durante as demonstrações e apresentações serão mostrados trechos de código Java e XML*
- *Para tirar o maior proveito das informações apresentadas (e entender os benefícios que podem trazer), é desejável que você tenha*
 - *Experiência com desenvolvimento em Java (você deve estar familiarizado com a estrutura de código Java)*
 - *Conhecimento elementar de XML*
 - *Noções mínimas de J2EE (componentes de um EJB e de uma aplicação Web com servlets e JSP)*

12 práticas fundamentais do XP

SUCESU-SP 2002

- *Jogo do planejamento*
- *Refinamento do design*
- *Ritmo saudável*
- *Design simples*
- **Testes**
- **Integração contínua**
- *Programação em duplas*
- *Participação do cliente*
- *Lançamentos pequenos*
- *Posse coletiva*
- *Metáfora*
- *Padrões de codificação*



- *O que são testes automáticos?*
 - São programas que avaliam se outro programa funciona como esperado e retornam resposta 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
- *O que é integração?*
 - Montagem de todo o sistema (compilação, ligação com dependências, configuração, empacotamento, testes)
 - Integração deve resultar em uma **versão funcional**
 - Mesmo que todas as unidades estejam funcionando, integração pode falhar: é preciso testar a integração!
 - + complicada quando código é desenvolvido em equipe

- *Por que não?*
 - Como saber se o recurso funciona sem testar?
 - Como saber se **ainda** funciona após refatoramento?
- *Testes dão maior segurança: **coragem** para mudar*
 - Que adianta a OO isolar a interface da implementação se programador tem **medo** de mudar a implementação?
 - Código testado é mais **confiável**
 - Código testado **pode ser alterado** sem medo
- *Como saber quando o projeto está pronto*
 - Testes == requisitos 'executáveis'
 - Testes de unidade devem ser executados o tempo todo
 - Escreva os testes **antes**. Quando todos rodarem 100%, o projeto está concluído!

Por que integrar continuamente?

- *"Reduz o tempo passado no inferno da integração" [8]*
 - *Quanto mais tempo durarem os bugs de integração, mais difíceis são de eliminar*
- *Integração contínua expõe o estado atual do desenvolvimento permanentemente*
 - *Permite avaliar e reavaliar prazos*
 - *Permite encontrar problemas de design rapidamente*
 - *Permite executar testes funcionais e de aceitação a qualquer momento*
 - *Estimula pequenos lançamentos e design simples*
- *Quando integrar?*
 - *Pelo menos uma vez por dia (sistemas grandes) ou +*

Como implementar integração contínua?

SUCESU-SP 2002

- *É possível usando ferramentas de código-fonte aberto*
 - **CVS, Ant, JUnit** e extensões para essas ferramentas
 - São ferramentas estáveis, de qualidade e largamente utilizadas (inclusive em grandes projetos)
 - Cumprem os requisitos mínimos necessários para viabilizar a integração contínua
- Segundo Fowler [8] os requisitos essenciais para implementar a integração contínua são
 - 1. Ter um único lugar de onde possam ser obtidas as fontes mais recentes
 - 2. Ter um único comando para montar a aplicação a partir das fontes do repositório
 - 3. Ter um único comando para rodar todos os testes

Parte I: automação de **testes de unidade**

- **JUnit**
- Extensões do JUnit: DBUnit, J2EEUnit, Mock Objects
- Testes de performance: JUnitPerf e JMeter

Parte II: automação do processo de **construção** (build)

- Jakarta **Ant**

Parte III: automação de **testes em aplicações Web**

- Jakarta **Cactus** e HttpUnit

Parte IV: automação da **integração contínua**

- Controle de versões: **CVS**
- Comparação e demonstração de ferramentas: CruiseControl, AntHill e Jakarta Gump

Código-fonte usado nas demonstrações estará disponível para download!

COMDEX

SUCESU-SP 2002

*Implementando
XP em Java*

parte 1

JU

Automação de

Testes de Unidade

com ***JUnit*** e extensões

www.junit.org

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

- 'Padrão' para **testes de unidade** em Java
 - Desenvolvido por Kent Beck (o guru do XP) e Erich Gamma (o G do GoF "Design Patterns")
- Testar é bom mas é chato; JUnit torna as coisas mais agradáveis, facilitando
 - 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

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

```
import junit.framework.*;  
class SuaClasseTest extends TestCase {...}
```
- Para cada método *xxx(args)* a ser testado defina um método *public void testXxx()* no test case
 - SuaClasse:

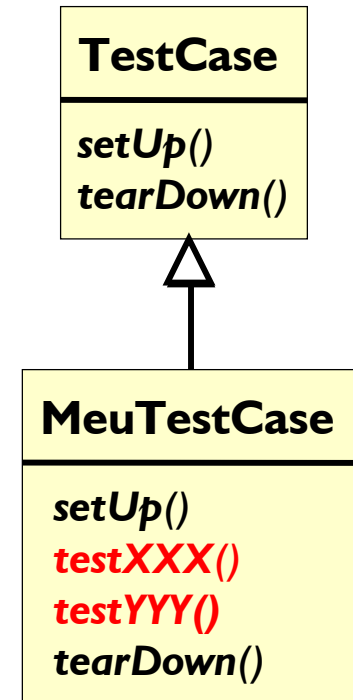
```
public boolean equals(Object o) { ... }
```
 - SuaClasseTest:

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

- 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());  
    }  
}
```

- 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**



```
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') {
                ; /* do nothing */
            } else {
                buffer.append( (char)c );
            }
        }
        return buffer.toString();
    }
}
```

veja demonstração

junitdemo.zip

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);
    }
}
```

Construtor precisa ser publico, receber String name e chamar super(String name) *

Método começa com "test" e é sempre public void

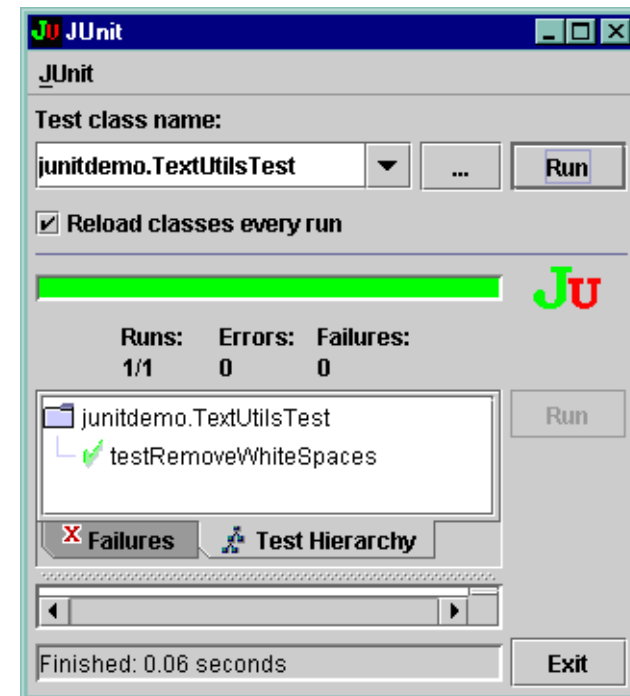
* Pode ser dispensado usando patch (veja www.junit.org)

veja demonstração

junitdemo.zip

- 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>`
 - tarefa do Apache Ant
- Ou forneça um `main()`:

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



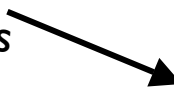
- Permite executar uma coleção de testes
 - Método **addTest(TestSuite)** adiciona um teste na lista
- Padrão de codificação (usando reflection):
 - retornar um TestSuite em cada test-case:

```
public static TestSuite 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;  
    }  
}
```

Pode incluir
outras suites

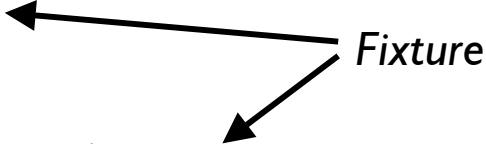


veja demonstração

junitdemo.zip

- 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

veja exemplo

jxunitdemo.zip

- *É tão importante testar o cenário de falha do seu código 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(); // no entities to resolve!
    try {
        // Following method call must cause exception!
        ParameterEntityTag tag = parser.resolveEntity("bogus");
        fail("Should have caused EntityNotFoundException!");
    } catch (EntityNotFoundException e) {
        // success: exception occurred as expected
    }
}
```

Afirmações do J2SDK1.4 (assertions)

- São expressões booleanas que o programador define para afirmar uma condição que ele acredita ser verdade
 - 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)
 - Melhoram a qualidade do código: tipo de teste
 - Devem ser usadas durante o desenvolvimento e desligadas na produção (afeta a performance)
 - Não devem ser usadas como parte da lógica do código
- Afirmações são um recurso novo do JSDK1.4.0
 - Nova palavra-chave: **assert**
 - 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`

- 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!";
}
```

- Provocam um **AssertionError** quando falham (que 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 J2SDK 1.4 e JUnit são complementares
 - JUnit testa a interface dos métodos
 - `assert` testa trechos de lógica dentro dos métodos

veja exemplo

junitdemo.zip

- **Acesso aos dados de métodos sob teste**
 - Métodos **private** e variáveis locais não podem ser testadas com JUnit.
 - Dados devem ser pelo menos **package-private** (friendly)
- **Soluções com refatoramento**
 - Isolar em métodos **private** apenas código inquebrável
 - Transformar métodos **private** 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 **private**
 - <http://www.extreme-java.de/junitx/index.html> veja exemplo

Boas práticas: metodologia "test-first"

- Testes geralmente ...
 - ... são mais simples que código a ser testado
 - ... refletem com clareza o que se espera do código, portanto, devem ser escritos **antes** do código!
 - Testes definem com precisão o que precisa ser feito
 - Estabelece uma meta clara para cada unidade de código
 - Evita que se perca tempo desenvolvendo o que é desnecessário
 - Desenvolvimento usando metodologia "**test-first**"
 1. Escreva o **esqueleto da sua classe** (métodos vazios)
 2. Escreva a **sua classe de teste** e implemente todos os testes (um para cada método ou condição importante)
 3. **Rode os testes**. Todos os testes devem falhar
 4. **Implemente** uma unidade de código e rode os testes
- ➔ Quando todos os testes rodarem com sucesso a sua classe está pronta!

COMDEX

Como escrever bons testes

SUCESU-SP 2002

- *JUnit facilita bastante a criação e execução de testes, mas elaborar bons testes exige mais*
 - *O que testar? Como saber se testes estão completos?*
- *"Teste tudo o que pode falhar" [2]*
 - *Métodos triviais (get/set) não precisam ser testados.*
 - *Será? E se houver uma rotina de validação no método set?*
 - *Métodos get/set **bem feitos** não falham (não devem conter lógica)!*
- *É melhor ter testes a mais que testes a menos*
 - *Use **assertNotNull()** sempre que puder (reduz drasticamente erros de NullPointerException difíceis de encontrar)*
 - *Reescreva seu código para que fique mais fácil de testar*
 - *Escreva um teste para cada afirmação **assertXXX()***
- *Bugs revelam testes*
 - *Achou um bug? Não conserte sem antes escrever um teste que o pegue (se você não o fizer, ele volta)!*

Como lidar com testes difíceis

SUCESU-SP 2002

- **Testes devem ser simples e suficientes**
 - **XP**: design mais simples que resolva o problema; sempre pode-se escrever novos testes, quando necessário
- **Não complique**
 - Não teste o que é **responsabilidade** de outra classe/método
 - **Assuma** que outras classes e métodos funcionam
- **Testes difíceis (ou que parecem difíceis)**
 - Aplicações gráficas: eventos, layouts, threads
 - Objetos inacessíveis, métodos privados, Singletons
 - Objetos que dependem de outros objetos
 - Objetos cujo estado varia devido a fatores imprevisíveis
- **Soluções**
 - **Alterar o design** da aplicação para facilitar os testes
 - **Simular** dependências usando proxies e stubs

- *O que testar?* [11]
 - Assumir que GUI (Swing, AWT, etc.) funciona
 - Concentrar-se na lógica de negócio e não na UI
- *Como testar?*
 - "Emagrecer" o código para **reduzir a chance de falha**
 - Usar **MVC**: isolar lógica de apresentação e controle
 - **Separar** código confiável do código que pode falhar
 - Usar **mediadores** (Mediator pattern) para intermediar interações entre componentes
- *Exemplo: event handlers*
 - Devem ter 0% de lógica de negócio: "A Stupid GUI is an Unbreakable GUI" (Robert Koss, Object Mentor) [9]
 - Responsabilidades delegadas a mediadores testáveis

Como lidar com Singletons

- *JUnit sempre reinicia os objetos ao chamar cada método*
 - *Exceção: singletons - cópia retornada é mesma usada anteriormente*
 - *Problema: fixtures podem não ser reinicializados para cada teste!*
- *Alternativas para solucionar o problema*
 - *Criar um método público na classe original que reinicialize o Singleton, e chamar esse método no **tearDown()***

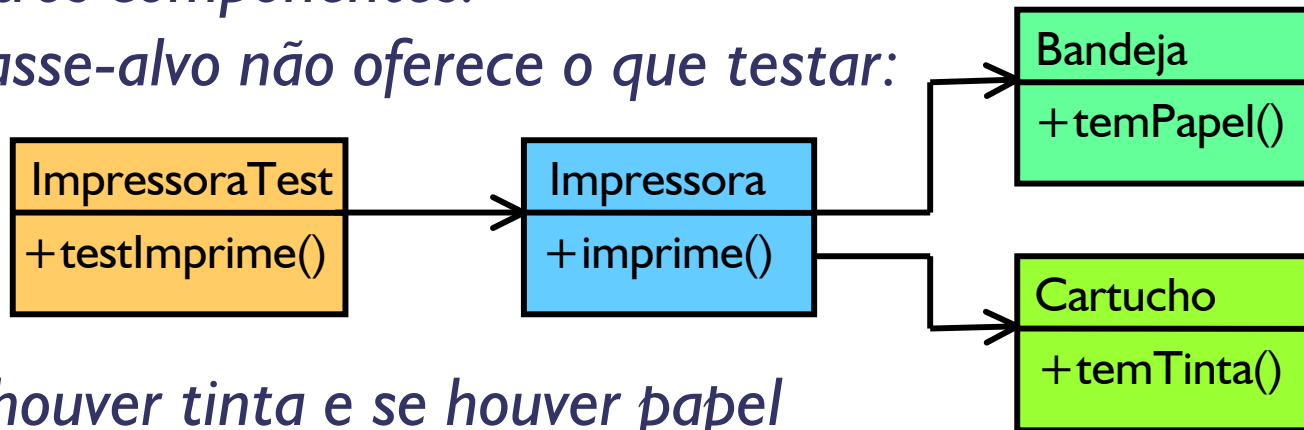
```
private HighLander() { ... } HighLander.class  
private static HighLander instance = null;  
public synchronized static HighLander getInstance() {  
    if (instance == null) { instance = new HighLander(); }  
    return instance;  
}  
  
static void resetSingleton(HighLanderTest testCase) {  
    if (testCase != null) {  
        instance = null; }  
}
```

Chame `resetSingleton(this)` no `tearDown()` de `HighLanderTest`

- *Usar o JUnitX para ter acesso à referência **instance***

- **Problema**

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



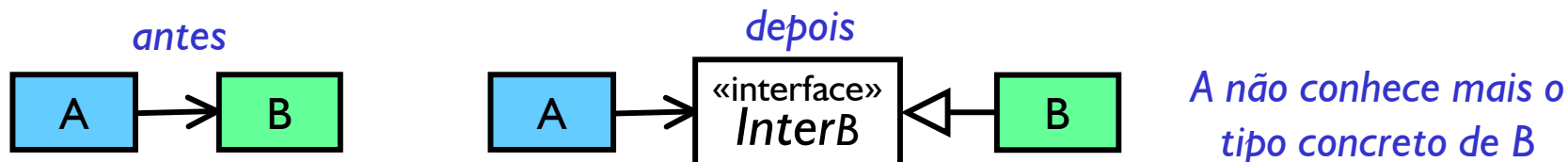
- Se houver tinta e se houver papel método `void imprime()` deve funcionar
- Como saber se há ou não tinta e papel?

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

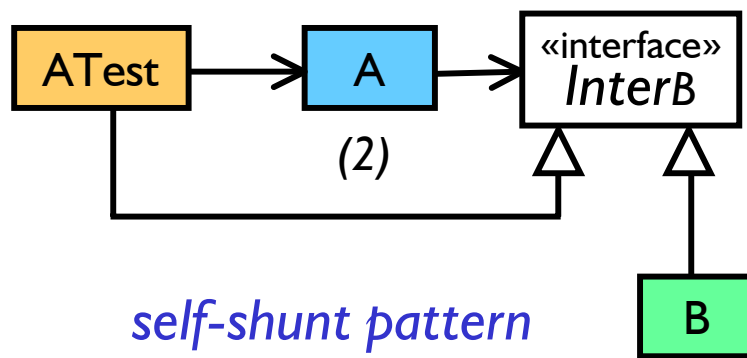
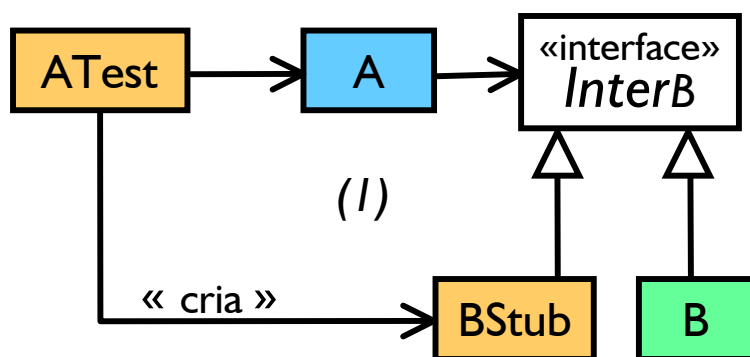
COMDEX SUCESU-SP 2002

Stubs: objetos "impostores"

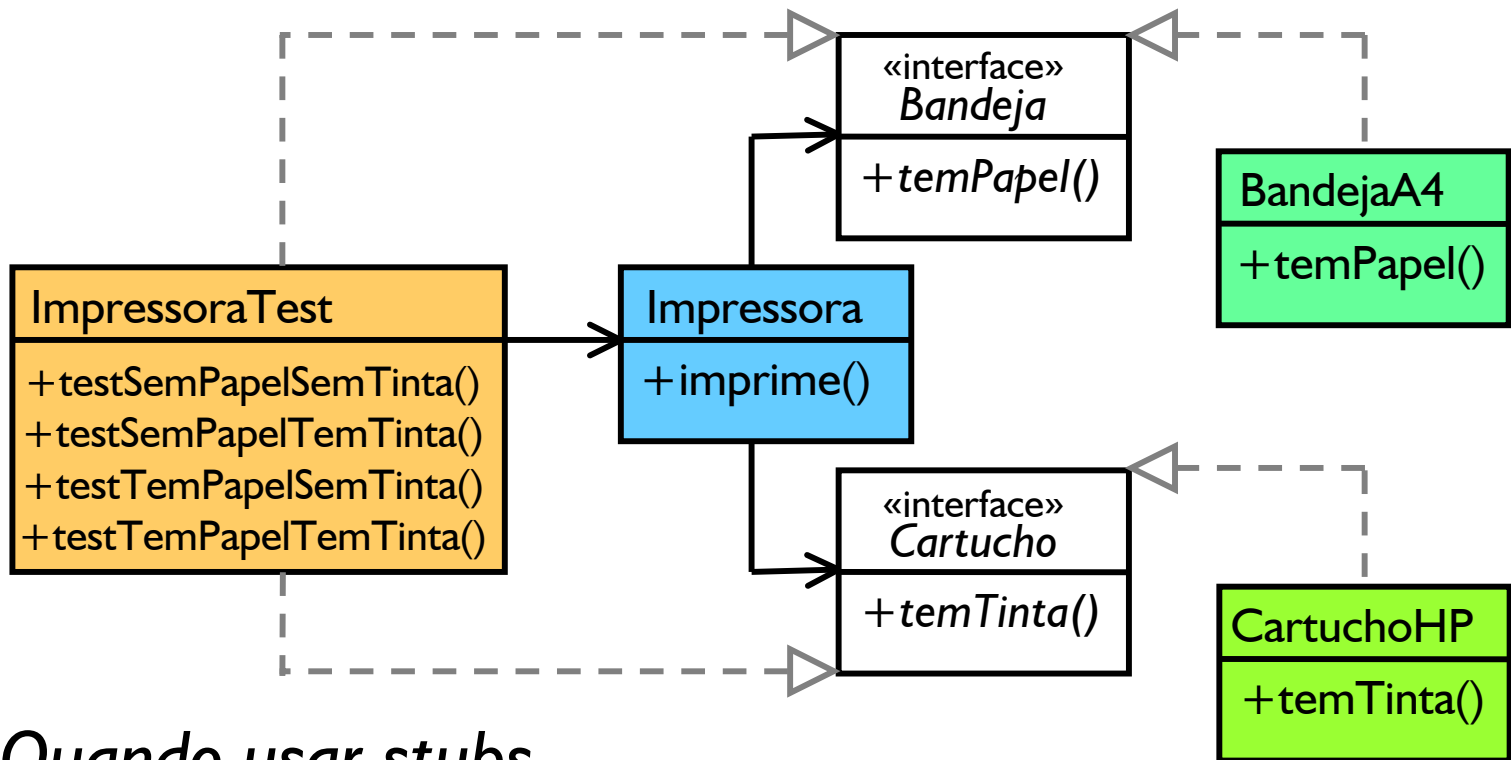
- É possível remover dependências de código-fonte refatorando o código para usar interfaces



- 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 não existe ainda (não está pronta)*
 - *Dependências tem 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 investimento
 - Use soluções open-source prontas!
 - **DBUnit**: extensão do JUnit para testar aplicações JDBC
<http://dbunit.sourceforge.net> veja exemplo **dbunitdemo.zip**
 - **JUnitEE**: extensão do JUnit para testar aplicações J2EE
e <http://junit.ee.sourceforge.net> veja exemplo **junitdemo.zip**
- Usar **proxies** (mediadores) para serviços reais
 - Testa a **integração** real do componente com seu ambiente
 - Solução open-source:
 - **Cactus**: testa integração com servlet containers

- **Mock objects** (MO) é uma estratégia similar ao uso de stubs mas que **não implementa nenhuma lógica**
 - Um mock object não é, portanto, um stub, pois não simula o funcionamento do objeto em qualquer situação
- Comportamento é controlado pela classe de teste que
 - Define comportamento esperado (valores retornados, etc.)
 - Passa MO configurado para objeto a ser testado
 - Chama métodos do objeto (que usam o MO)
- Implementações open-source que facilitam uso de MOs
 - **EasyMock** (tammofreese.de/easymock/) e **MockMaker** (www.xpdeveloper.com) geram MOs a partir de interfaces
 - **Projeto MO** (mockobjects.sourceforge.net) coleção de mock objects e utilitários para usá-los

- Às vezes testes são difíceis por causa de limitações do próprio paradigma orientado a objetos
 - Encapsulamento: impede acesso além da interface
 - Herança e polimorfismo podem ser insuficientes para modelar eficientemente condição de teste mais abrangente
- Soluções:
 - Design patterns + programação genérica (reflection)
 - Extensões ao paradigma OO: Aspect-Oriented, Subject-Oriented ou Adaptive programming
- **AspectJ** estende Java com recursos "Aspect-Oriented"
 - Linguagem onde se pode representar requisitos como "aspectos" que envolvem classes/métodos não-relacionados
 - **Simplifica** criação de testes que envolvem várias classes

- **Caso específico: resposta de servidores Web**
 - 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 tipo "caixa-preta"
- **Soluções (extensões do JUnit)**
 - **HttpUnit** e **ServletUnit**:
 - permite 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
 - Onde encontrar: (httpunit | jxunit | xmlunit).sourceforge.net

veja exemplo

xmlunitdemo.zip

- **JUnitPerf** (www.clarkware.com)
 - Coleção de decoradores para medir performance e escalabilidade em testes JUnit **existentes**
- **TimedTest**
 - Executa um teste e mede o **tempo** transcorrido
 - Define um tempo máximo para a execução. Teste falha se execução durar mais que o tempo estabelecido
- **LoadTest**
 - Executa um teste com uma **carga** simulada
 - Utiliza timers para distribuir as cargas usando distribuições randômicas
 - Combinado com **TimerTest** para medir tempo com carga
- **ThreadedTest**
 - Executa o teste em um thread separado

veja demonstração

[junitperfdemo.zip](#)

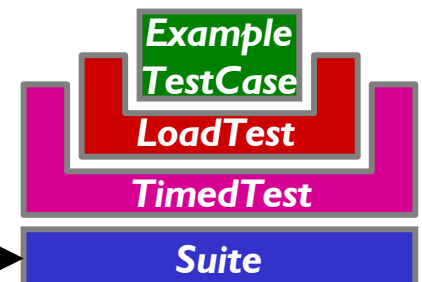
```
import com.clarkware.junitperf.*;
import junit.framework.*;
```

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

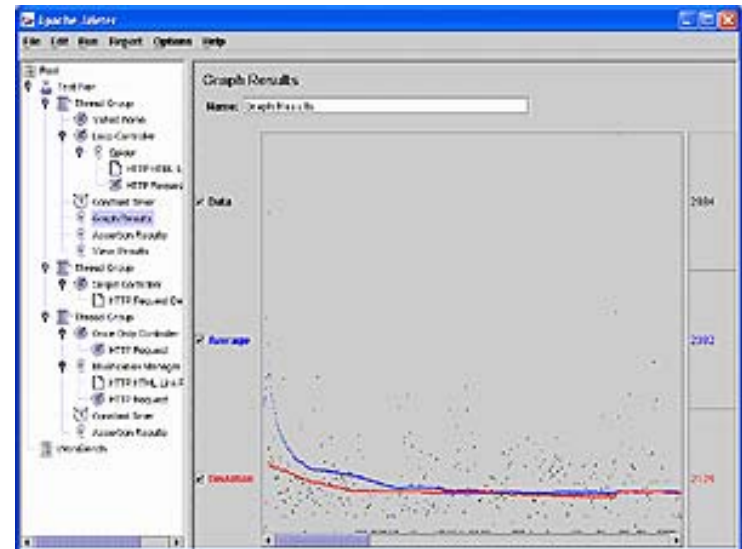
TestCase existente

Decorador de carga

Decorador de tempo



- Uma ótima ferramenta open-source para testar os limites de uma aplicação é o Apache JMeter
 - <http://jakarta.apache.org/jmeter>
 - Simula carga pesada em servidor, rede ou objeto
 - Testa performance em aplicações Web, bancos de dados, sistema 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

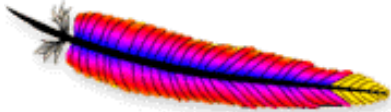


- *É possível desenvolver software de qualidade com um investimento mínimo em ferramentas*
 - *Há ótimas opções de ferramentas open-source*
 - *Ferramentas tornam mais fácil a adoção de práticas como "test-first" e "refactoring" que aumentam a qualidade do software*
- *JUnit é muito simples, é de graça, mas vale muito!*
 - *"Never in the field of software development was so much owed by so many to so few lines of code" Martin Fowler, sobre o JUnit*
 - *Principais ferramentas comerciais incluem o JUnit: Together Control Center, Sun Forté for Java, IBM Websphere Studio, etc.*
- *Vale a pena investir tempo para desenvolver e aperfeiçoar a prática constante de escrever testes com o JUnit*
 - *mais produtividade, maior integração de equipes*
 - *produtos de melhor qualidade, com prazo previsível*
 - *menos stress, mais organização*

COMDEX

SUCESU-SP 2002

*Implementando
XP em Java*



The Jakarta Project
<http://jakarta.apache.org>

parte 2



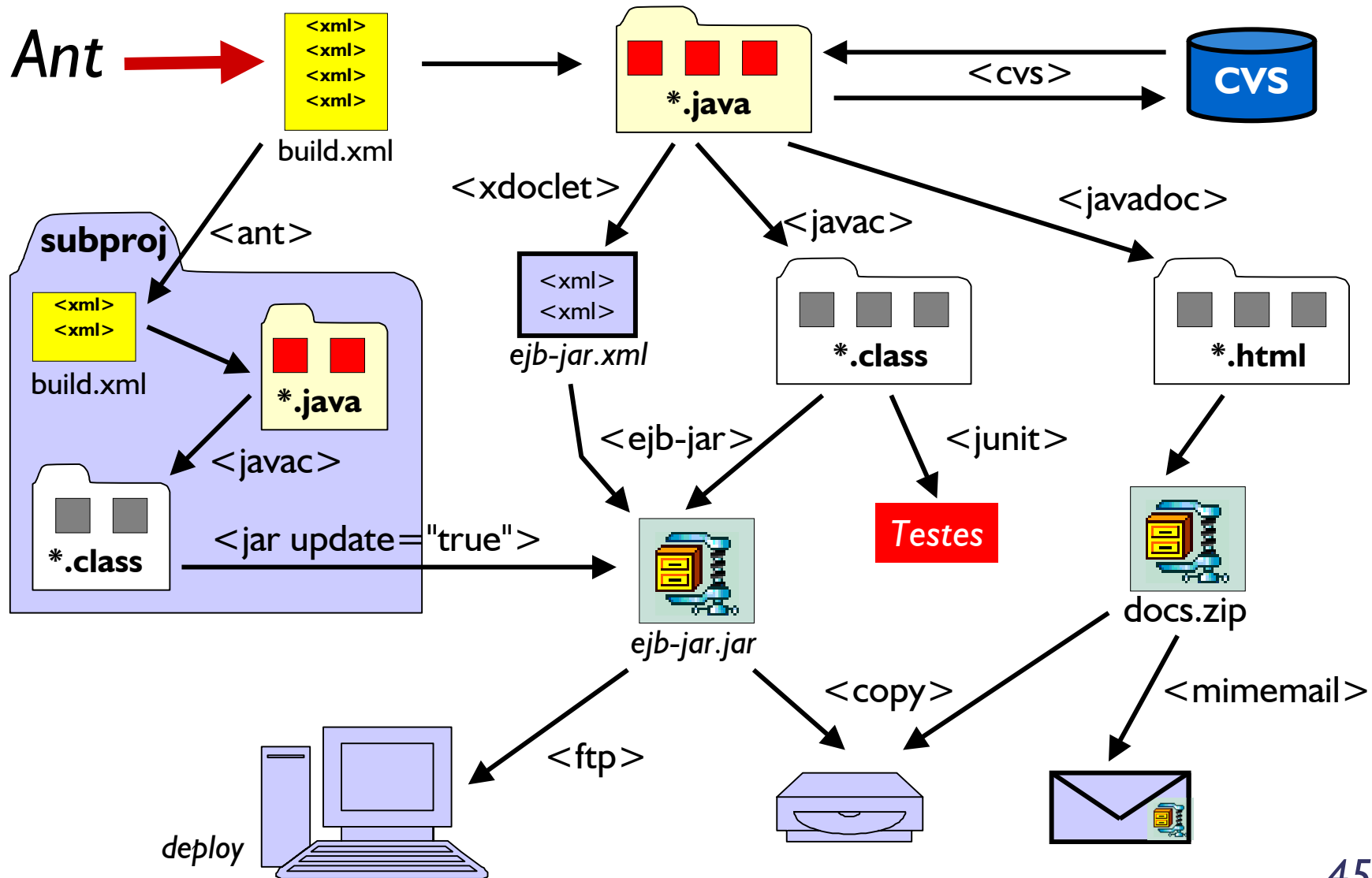
*Construção de aplicações com **Apache Ant***

jakarta.apache.org/ant/

- *Ferramenta para construção de aplicações*
 - *Implementada em Java*
 - *Baseada em roteiros XML*
 - *Extensível (via scripts ou classes)*
 - *'padrão' do mercado*
 - *Open Source (Grupo Apache, Projeto Jakarta)*
- *Semelhante a **make**, porém*
 - *Mais simples e estruturada (XML)*
 - *Mais adequada a tarefas comuns em projetos Java*
 - *Independente de plataforma*

- Para montar praticamente **qualquer** aplicação Java que consista de mais que meia dúzia de classes;
Aplicações
 - *distribuídas em pacotes*
 - *que requerem a definição de classpaths locais, e precisam vincular código a bibliotecas (JARs)*
 - *cuja criação/instalação depende de mais que uma simples chamada ao javac. Ex: RMI, CORBA, EJB, servlets, JSP,...*
- Para automatizar processos frequentes
 - *Javadoc, XSLT, implantação de serviços Web e J2EE (deployment), CVS, criação de JARs, testes, FTP, email*

- *Ant executa roteiros escritos em XML: 'buildfiles'*
- *Cada projeto do Ant possui um buildfile*
 - *subprojetos podem ter, opcionalmente, buildfiles adicionais chamados durante a execução do primeiro*
- *Cada projeto possui uma coleção de alvos*
- *Cada alvo consiste de uma seqüência de tarefas*
- *Exemplos de execução*
 - ▶ **ant**
 - *procura build.xml no diretório atual e roda alvo default*
 - ▶ **ant -buildfile outro.xml**
 - *executa alvo default de arquivo outro.xml*
 - ▶ **ant compile**
 - *roda alvo 'compile' e possíveis dependências em build.xml*



- O buildfile é um arquivo XML: **build.xml** (default)
- Principais elementos

<project default="alvo_default">

- Elemento raiz (obrigatório): define o projeto.

<target name="nome_do_alvo">

- Coleção de tarefas a serem executadas em seqüência
- Deve haver pelo menos um <target>

<property name="nome" value="valor">

- **pares nome/valor** usados em atributos dos elementos do build.xml da forma `${nome}`
- **propriedades** também podem ser definidas em linha de comando (`-Dnome=valor`) ou lidas de arquivos externos (atributo **file**)
- **tarefas (mais de 130) - dentro dos alvos.**
 - `<javac>`, `<jar>`, `<java>`, `<copy>`, `<mkdir>`, ...

```

<?xml version="1.0" encoding="iso-8859-1" ?>
<!-- Compila diversos arquivos .java -->
<project default="compile" basedir=".>
  <property name="src.dir" value="${basedir}/src" />
  <property name="build.dir" value="build" />
  <target name="init">
    <echo> Criando diretório </echo>
    <mkdir dir="${build.dir}" />
  </target>
  <target name="compile" depends="init"
    description="Compila os arquivos-fonte">
    <javac srcdir="${src.dir}" destdir="${build.dir}">
      <classpath>
        <pathelement location="${build.dir}" />
      </classpath>
    </javac>
  </target>
</project>

```

Propriedades

Alvos

Tarefas

- Executando buildfile da página anterior

```
C:\usr\palestra\antdemo> ant
Buildfile: build.xml

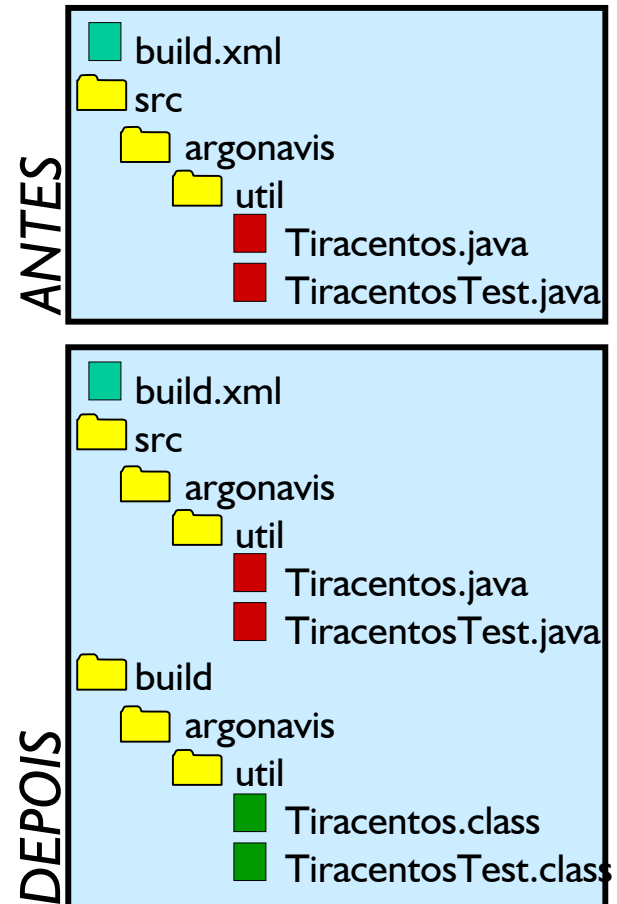
init:
  [echo] Criando diretório
  [mkdir] Created dir:
  C:\usr\palestra\antdemo\build

compile:
  [javac] Compiling 2 source files to
  C:\usr\palestra\antdemo\build

BUILD SUCCESSFUL
Total time: 4 seconds
C:\usr\palestra\antdemo>
```

veja demonstração

[antdemo.zip](#)



O que se pode fazer com Ant?

SUCESU-SP 2002

- **Compilar.**
`<javac>`, `<csc>`
- **Gerar documentação**
`<javadoc>`, `<junitreport>`,
`<style>`, `<stylebook>`
- **Gerar código (XDoclet)**
`<ejbdoclet>`, `<webdoclet>`
- **Executar programas**
`<java>`, `<apply>`, `<exec>`
`<ant>`, `<sql>`
- **Empacotar e comprimir**
`<jar>`, `<zip>`, `<tar>`,
`<war>`, `<ear>`, `<cab>`
- **Expandir, copiar, instalar**
`<copy>`, `<delete>`, `<mkdir>`,
`<unjar>`, `<unwar>`, `<unzip>`
- **Acesso remoto**
`<ftp>`, `<telnet>`, `<cvs>`,
`<mail>`, `<mimemail>`
- **Montar componentes**
`<ejbc>`, `<ejb-jar>`, `<rmic>`
- **Testar unidades de código**
`<junit>`
- **Criar novas tarefas**
`<taskdef>`
- **Executar roteiros e sons**
`<script>`, `<sound>`

- **<javac>**: Chama o compilador Java

```
<javac srcdir="dirfontes" destdir="dirbuild" >  
  <classpath>  
    <pathelement path="arquivo.jar" />  
    <pathelement path="/arquivos" />  
  </classpath>  
  <classpath idref="extra" />  
</javac>
```

- **<jar>**: Monta um JAR

```
<jar destfile="bin/programa.jar">  
  <manifest>  
    <attribute name="Main-class"  
              value="exemplo.main.Exec">  
  </manifest>  
  <fileset dir="${build.dir}" />  
</jar>
```

veja exemplos

hellojsp.zip

Tarefas do sistema de arquivos

SUCESU-SP 2002

- **<mkdir>**: *cria diretórios*
`<mkdir dir="diretorio" />`
- **<copy>**: *copia arquivos*
`<copy todir="dir" file="arquivo" />`
`<copy todir="dir">`
 `<fileset dir="fonte"`
 `includes="*.txt" />`
`</copy>`
- **<delete>**: *apaga arquivos*
`<delete file="arquivo" />`
`<delete dir="diretorio" />`

- **<javadoc>**: Gera documentação do código-fonte.
 - Alvo abaixo gera documentação e exclui classes que contém 'Test.java'

```
<target name="generate-docs">
  <mkdir dir="docs/api" />
  <copy todir="tmp">
    <fileset dir="{src.dir}">
      <include name="**/*.java" />
      <exclude name="**/**Test.java" />
    </fileset>
  </copy>
  <javadoc destdir="docs/api"
    packagenames="argonavis.*"
    sourcepath="tmp" />
  <delete dir="tmp" />
</target>
```

- Podem ser definidas com **<property>**

```
<property name="app.nome" value="jmovie" />
```

- Podem ser carregadas de um arquivo

```
<property file="c:/conf/arquivo.conf" />
```

```
app.ver=1.0  
docs.dir=c:\docs\  
codigo=15323
```

arquivo.conf

- Podem ser passadas na linha de comando

```
c:\> ant -Dautor=Wilde
```

- Para recuperar o valor, usa-se **\${nome}**

```
<jar destfile="${app.nome}-${app.ver}.jar" />
```

```
<echo message="O autor é ${autor}" />
```

```
<mkdir dir="build${codigo}" />
```

- **<tstamp>**: Grava um instante
 - A hora e data podem ser recuperados como propriedades
 - `#{TSTAMP}` `hhmm` `1345`
 - `#{DSTAMP}` `aaaammdd` `20020525`
 - `#{TODAY}` `dia mes ano` `25 May 2002`
 - Novas propriedades podem ser definidas, locale, etc.
 - Uso típico: `<tstamp />`
- **<property environment="env">**: Propriedade de onde se pode ler variáveis de ambiente do sistema
 - Dependente de plataforma

```
<target name="init">
  <property environment="env" />
  <property name="j2ee.home"
            value="env.J2EE_HOME" />
</target>
```

veja exemplos

minied.zip

Tipos de dados: arquivos e diretórios

- **<fileset>**: árvore de arquivos e diretórios
 - Conteúdo do conjunto pode ser reduzido utilizando elementos `<include>` e `<exclude>`
 - Usando dentro de tarefas que manipulam com arquivos e diretórios como `<copy>`, `<zip>`, etc.

```
<copy todir="${build.dir}/META-INF">
  <fileset dir="${xml.dir}" includes="ejb-jar.xml"/>
  <fileset dir="${xml.dir}/jboss">
    <include name="*.xml" />
    <exclude name="*-orig.xml" />
  </fileset>
</copy>
```

- **<dirset>**: árvore de diretórios
 - Não inclui arquivos individuais

- **<patternset>**: representa coleção de padrões

```
<patternset id="project.jars" >  
  <include name="**/*.jar"/>  
  <exclude name="**/*-test.jar"/>  
</patternset>
```

- **<path>**: representa uma coleção de caminhos
 - Associa um ID a grupo de arquivos ou caminhos

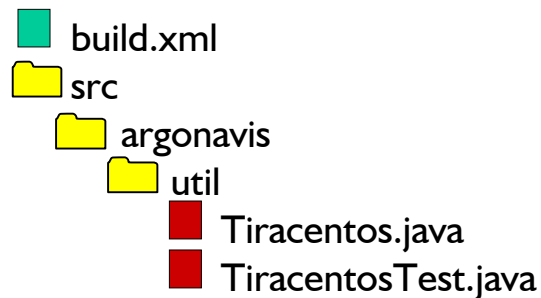
```
<path id="server.path">  
  <pathelement path="${j2ee.home}/lib/locale" />  
  <fileset dir="${j2ee.home}/lib">  
    <patternset refid="project.jars" />  
  </fileset>
```

```
</path>
```

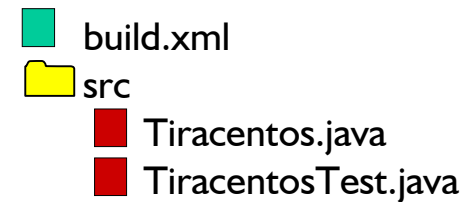
```
<target name="compile" depends="init">  
  <javac destdir="${build.dir}" srcdir="${src.dir}">  
    <classpath refid="server.path" />  
  </javac>  
</target>
```


Tipos de dados: File Mapper

- **<mapper>**: altera nomes de arquivos durante cópias ou transformações
 - Seis tipos: *identity*, *flatten*, *merge*, *regex*, *glob*, *package*



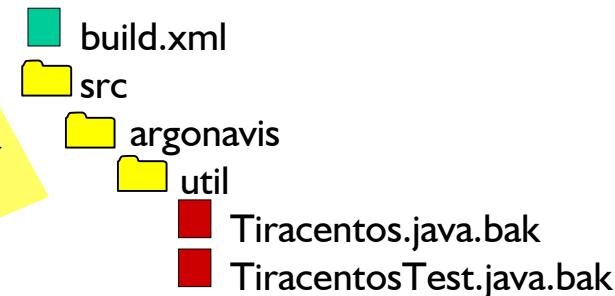
```
<mapper type="flatten" />
```



```
<mapper type="package" from="*.java" to="*.txt"/>
```



```
<mapper type="glob" from="*.java" to="*.java.bak"/>
```



- Permitem a seleção dos elementos de um fileset usando critérios além dos definidos por `<include>` e `<exclude>`
- Sete seletores básicos (pode-se criar novos)
 - **`<contains>`** - Seleciona arquivos que contém determinado texto
 - **`<date>`** - Arquivos modificados antes ou depois de certa data
 - **`<depend>`** - Seleciona arquivos cuja data de modificação seja posterior a arquivos localizados em outro lugar
 - **`<depth>`** - Seleciona arquivos encontrados até certa profundidade de uma árvore de diretórios
 - **`<filename>`** - Equivalente ao `include` e `exclude`
 - **`<present>`** - Seleciona arquivo com base na sua (in)existência
 - **`<size>`** - Seleciona com base no tamanho em bytes

Exemplo: Seleciona arquivos do diretório "fonte" que também estão presentes em "destino"

```
<fileset dir="fonte">  
  <present targetdir="destino"/>  
</fileset>
```

- **<filter>** e **<filterset>**: Permite a substituição de padrões em arquivos durante a execução de uma tarefa
 - Caractere default: @
- **Exemplo**: a tarefa abaixo irá substituir todas as ocorrências de @javahome@ por c:\j2sdk1.4.0 nos arquivos copiados

```
<copy todir="${dest.dir}">  
  <fileset dir="${src.dir}"/>  
  <filterset>  
    <filter token="javahome" value="c:\j2sdk1.4.0"/>  
  </filterset>  
</copy>
```

- Pares **token=valor** podem ser carregados de arquivo:

```
<filterset>  
  <filtersfile file="build.properties" />  
</filterset>
```

```
<ear destfile="app.ear" appxml="application.xml">
  <fileset dir="${build}" includes="*.jar,*.war"/>
</ear>
```

```
<ejbjar srcdir="${build}" descriptor="description.xml" ... >
  <jboss destdir="${deployjars.dir}" />
</ejbjar>
```

Há suporte aos principais servidores

```
<war warfile="bookstore.war" webxml="meta/metainf.xml">
  <fileset dir="${build}/${bookstore2}" >
    <include name="*.tld" />
    <exclude name="web.xml" />
  </fileset>
  <classes dir="${build}" >
    <include name="database/*.class" />
  </classes>
  <lib dir="bibliotecas" />
  <webinf dir="jboss-web.xml" />
</war>
```

↑ WEB-INF/web.xml

← Raiz do WAR

← WEB-INF/classes

← WEB-INF/lib

← WEB-INF/

<ejbdoclet> e <webdoclet>: Geram código

- Requer JAR de xdoclet.sourceforge.net
- Ideal para **geração automática de arquivos de configuração** (web.xml, ejb-jar.xml, application.xml, taglibs, struts-config, etc.) e **código-fonte** (beans, value-objects)

```
<ejbdoclet sourcepath="src" destdir="${build.dir}"
           classpathref="xdoclet.path" ejbspec="2.0">
  <fileset dir="src">
    <include name="**/*Bean.java" />
  </fileset>
  <remoteinterface/>
  <homeinterface/>
  <utilobject/>
  <entitypk/>
  <entitycmp/>
  <deploymentdescriptor destdir="${dd.dir}"/>
  <jboss datasource="java:/OracleDS" />
</ejbdoclet>
```

XDoclet faz muito mais que isto!

Detalhes da configuração do componente estão em arquivos de template externos

- **<java>**: roda o interpretador Java

```
<target name="runrmiclient">  
  <java classname="hello.rmi.HelloClient" fork="true">  
    <jvmarg value="-Djava.security.policy=rmi.policy"/>  
    <arg name="host" value="{remote.host}" />  
    <classpath refid="app.path" />  
  </java>  
</target>
```

- **<exec>**: executa um comando do sistema

```
<target name="orbd">  
  <exec executable="{java.home}\bin\orbd">  
    <arg line="-ORBInitialHost {nameserver.host}"/>  
  </exec>  
</target>
```

- **<apply>**: semelhante a <exec> mas usado em executáveis que operam sobre outros arquivos

- **<ftp>**: Realiza a comunicação com um servidor FTP remoto para upload ou download de arquivos
 - Tarefa opcional que requer NetComponents.jar (<http://www.savarese.org>)

```
<target name="remote.jboss.deploy" depends="dist">  
  <ftp server="${ftp.host}" port="${ftp.port}"  
    remotedir="/jboss/server/default/deploy"  
    userid="admin" password="jboss"  
    depends="yes" binary="yes">  
    <fileset dir="${basedir}">  
      <include name="*.war"/>  
      <include name="*.ear"/>  
      <include name="*.jar"/>  
    </fileset>  
  </ftp>  
</target>
```

veja demonstrações

ftpdemo.zip

maildemo.zip

- **<style>**: Transforma documentos XML em outros formatos usando folha de estilos XSLT (nativa)
 - Usa Trax (default), Xalan ou outro transformador XSL

```
<style basedir="xmldocs"
       destdir="htmldocs"
       style="xmltohtml.xsl" />
```

veja demonstração

styledemo.zip

- Elemento `<param>` passa valores para elementos `<xsl:param>` da folha de estilos

```
<style in="cartao.xml"
       out="cartao.html"
       style="cartao2html.xsl">
  <param name="docsdir"
         expression="/cartoes"/>
</style>
```

build.xml

cartao2html.xsl

```
(...)
<xsl:param name="docsdir"/>
(...)
<xsl:valueof select="$docsdir"/>
(...)
```


- **<sound>**: define um par de arquivos de som para soar no sucesso ou falha de um projeto
 - Tarefa opcional que requer Java Media Framework
- Exemplo:
 - No exemplo abaixo, o som frog.wav será tocado quando o build terminar sem erros fatais. Bark.wav tocará se houver algum erro que interrompa o processo:

```
<target name="init">  
  <sound>  
    <success source="C:/Media/frog.wav"/>  
    <fail source="C:/Media/Bark.wav"/>  
  </sound>  
</target>
```

- **<sql>**: Comunica-se com banco de dados através de driver JDBC

```
<property name="jdbc.url"
  value="jdbc:cloudscape:rmi://server:1099/Cloud" />
<target name="populate.table">
  <sql driver="COM.cloudscape.core.RmiJdbcDriver"
    url="{jdbc.url}"
    userid="helder"
    password="helder"
    onerror="continue">
    <transaction src="droptable.sql" />
    <transaction src="create.sql" />
    <transaction src="populate.sql" />
    <classpath refid="jdbc.driver.path" />
  </sql>
</target>
```

veja exemplo

hellocmp.zip

- Como o buildfile é um arquivo XML, pode-se incluir trechos de XML externos através do uso de *entidades externas*

sound.xml

```
<property file="sound.properties" />
<sound>
  <success source="{success.sound}"/>
  <fail source="{fail.sound}"/>
</sound>
```

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<!DOCTYPE project [
  <!ENTITY sound SYSTEM "sound.xml">
]>
<project default="dtd">
  <description>Gera um DTD para o Ant</description>
  <target name="init">
    &sound;
  </target>
  <target name="dtd" depends="init">
    <antstructure output="ant.dtd" />
  </target>
</project>
```

veja demonstração

soundemo.zip

- *Viabiliza a integração contínua:*
 - *Pode-se executar todos os testes após a integração com um único comando:*
 - **ant roda-testes**
- *Com as tarefas **<junit>** e **<junitreport>** é possível*
 - *executar todos os testes*
 - *gerar um relatório simples ou detalhado, em diversos formatos (XML, HTML, etc.)*
 - *executar testes de integração*
- *São tarefas opcionais. É preciso ter no \$ANT_HOME/lib*
 - *optional.jar (distribuído com Ant)*
 - *junit.jar (distribuído com JUnit)*

```
<target name="test" depends="build">
  <junit printsummary="true" dir="${build.dir}"
        fork="true">
    <formatter type="plain" usefile="false" />
    <classpath path="${build.dir}" /
    <test name="argonavis.dtd.AllTests" />
  </junit>
</target>
```

Formata os dados na tela (plain)
Roda apenas arquivo AllTests

```
<target name="batchtest" depends="build" >
  <junit dir="${build.dir}" fork="true">
    <formatter type="xml" usefile="true" />
    <classpath path="${build.dir}" />
    <batchtest todir="${test.report.dir}">
      <fileset dir="${src.dir}">
        <include name="**/*Test.java" />
        <exclude name="**/AllTests.java" />
      </fileset>
    </batchtest>
  </junit>
</target>
```

Gera arquivo XML
Inclui todos os arquivos que
terminam em TEST.java

- Gera um relatório detalhado (estilo JavaDoc) de todos os testes, sucessos, falhas, exceções, tempo, ...

```

<target name="test-report" depends="batchtest" >
  <junitreport todir="${test.report.dir}">
    <fileset dir="${test.report.dir}">
      <include name="TEST-*.xml" />
    </fileset>
    <report todir="${test.report.dir}/html"
            format="frames" />
  </junitreport>
</target>
  
```

Usa arquivos XML gerados por <formatter>

veja demonstração

junitdemo.zip

veja demonstração

dtdreader.zip

Unit Test Results

Designed for use with [JUnit](#) and [Ant](#).

Summary

Tests	Failures	Errors	Success rate	Time
122	4	0	96.72%	59.100

Note: failures are anticipated and checked for with assertions while errors are unanticipated.

Packages

Name	Tests	Errors	Failures	Time(s)
argonavis.dtd	70	0	4	43.070
argonavis.dtd.parsers	26	0	0	7.910
argonavis.dtd.taadata	26	0	0	8.120

- Há duas formas de estender o Ant com novas funções
 - Implementar roteiros usando JavaScript
 - Criar novas tarefas reutilizáveis
- A tarefa **<script>** permite embutir JavaScript em um buildfile. Pode-se
 - realizar operações aritméticas e booleanas
 - utilizar estruturas como **if/else**, **for**, **foreach** e **while**
 - manipular com os elementos do buildfile usando DOM
- A tarefa **<taskdef>** permite definir novas tarefas
 - tarefa deve ser implementada em Java e estender **Task**
 - método **execute()** contém código de ação da tarefa
 - cada atributo corresponde a um método **setXXX()**

veja demonstração

scriptdemo.zip

veja exemplos

foptask.zip

veja demonstração

taskdemo.zip

Integração com outras aplicações

- Ant provoca vários **eventos** que podem ser capturados por outras aplicações
 - Útil para implementar integração, enviar notificações por email, gravar logs, etc.
- **Eventos**
 - Build iniciou/terminou
 - Alvo iniciou/terminou
 - Tarefa iniciou/terminou
 - Mensagens logadas
- **Vários listeners e loggers pré-definidos**
 - Pode-se usar ou estender classe existente.
 - Para gravar processo (build) em XML:

```
ant -listener org.apache.tools.ant.XmlLogger
```

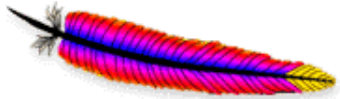

- *Produtos que integram com Ant e oferecem interface gráfica e eventos para buildfiles:*
 - **Antidote**: GUI para Ant (do projeto Jakarta)
 - <http://cvs.apache.org/viewcvs/jakarta-ant-antidote/>
 - **JBuilder** (AntRunner plug-in)
 - <http://www.dieter-bogdoll.de/java/AntRunner/>
 - **NetBeans** e **Forté for Java**
 - <http://ant.netbeans.org/>
 - **Visual Age** for Java (integração direta)
 - **JEdit** (AntFarm plug-in)
 - <http://www.jedit.org>
 - **Jext** (AntWork plug-in)
 - <ftp://jext.sourceforge.net/pub/jext/plugins/AntWork.zip>

- *Vale a pena aprender e ganhar experiência com o Ant*
 - *É bom, é de graça e todo mundo usa!*
 - *Facilita a compilação, depuração, execução, montagem, instalação, documentação e utilização de qualquer aplicação Java*
 - *Faz ainda transformação XSLT, geração de código e qualquer outra tarefa que o programador desejar*
 - *Você pode integrá-lo em sua aplicação. O código é aberto!*
 - *É mais fácil que make. É melhor que usar arquivos .bat e .sh*
 - *É independente de IDE e plataforma*
- *Use Ant mesmo que seu IDE já possua um "build"*
 - *Ant oferece muito mais recursos que qualquer comando "build" dos IDEs existentes hoje, é extensível e deixa seu projeto independente de um IDE específico*
 - *Os principais fabricantes de IDEs Java suportam Ant ou possuem plug-ins para integração com Ant*

COMDEX

SUCESU-SP 2002

*Implementando
XP em Java*



The Jakarta Project
<http://jakarta.apache.org>

parte 4

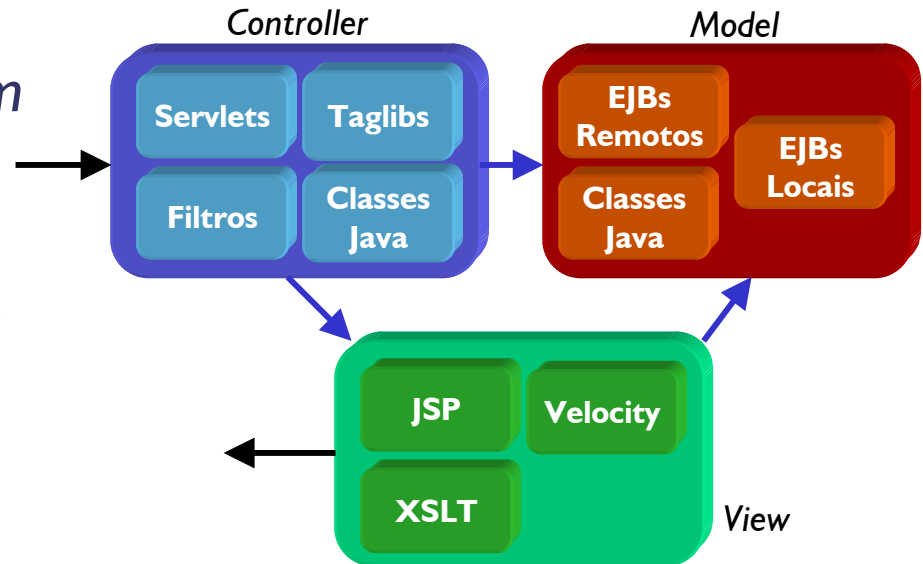


*Testes em aplicações Web com **Apache Cactus** e **HttpUnit***

jakarta.apache.org/cactus/
httpunit.sourceforge.net

- É um framework que oferece facilidades para testar componentes J2EE
 - Componentes Web (Camada de **C**ontrole)
 - Camada EJB (**M**odel) e cliente (**V**iew): indiretamente
- Produto Open Source do projeto Jakarta
 - Metas de curto prazo: testar componentes acima + EJB
 - Metas de longo prazo: oferecer facilidades para testar todos os componentes J2EE; ser o framework de referência para testes in-container.
- Cactus estende o JUnit framework
 - Execução dos testes é realizada de forma idêntica
 - TestCases são construídos sobre uma subclasse de `junit.framework.TestCase`

- Para testar aplicações que utilizam componentes J2EE
- Arquitetura MVC
 - Servlets, filtros e custom tags (**C**ontroladores)
 - JSPs (camada de apresentação: **V**iew, através de controladores)
 - EJB (**M**odelo de dados/ lógica de negócios)
- Cactus testa a integração desses componentes com seus containers
 - **não usa stubs** - usa o próprio container como servidor e usa JUnit como cliente
 - comunicação é intermediada por um **proxy**



- *Cactus utiliza os test cases simultaneamente no cliente e no servidor: **duas cópias***
 - *Uma cópia é instanciada pelo servlet container*
 - *Outra cópia é instanciada pelo JUnit*
- *Comunicação com o servlet container é feita através de um proxy (XXXRedirector)*
 - *JUnit envia requisições via HTTP para proxy*
 - *Proxy devolve resultado via HTTP e JUnit os mostra*
- *Há, atualmente (Cactus 1.3) três tipos de proxies:*
 - ***ServletRedirector**: para testar servlets*
 - ***JSPRedirector**: para testar JSP custom tags*
 - ***FilterRedirector**: para testar filtros de servlets*

- Parte da mesma classe (ServletTestCase) é executada no cliente, parte no servidor

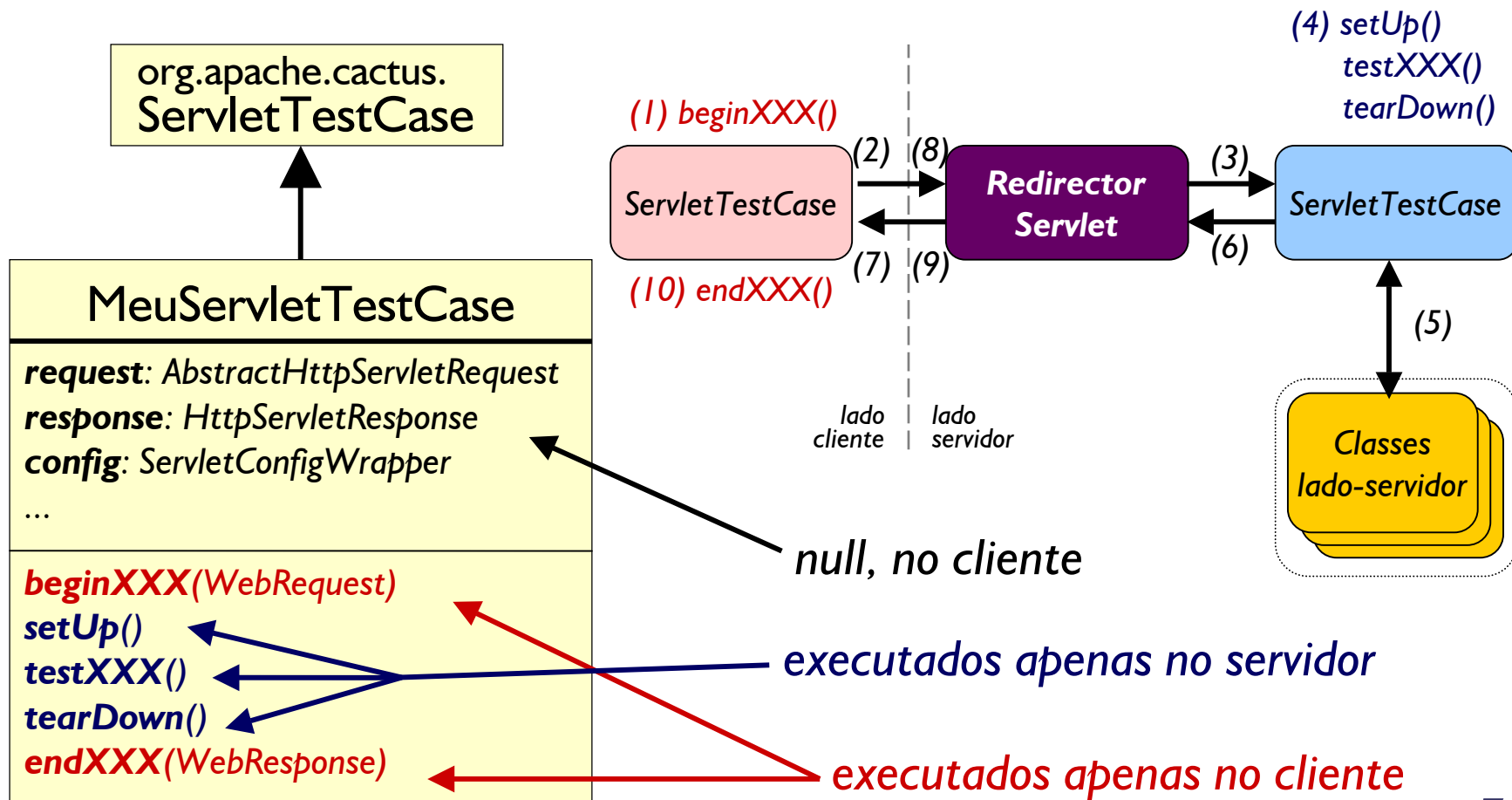
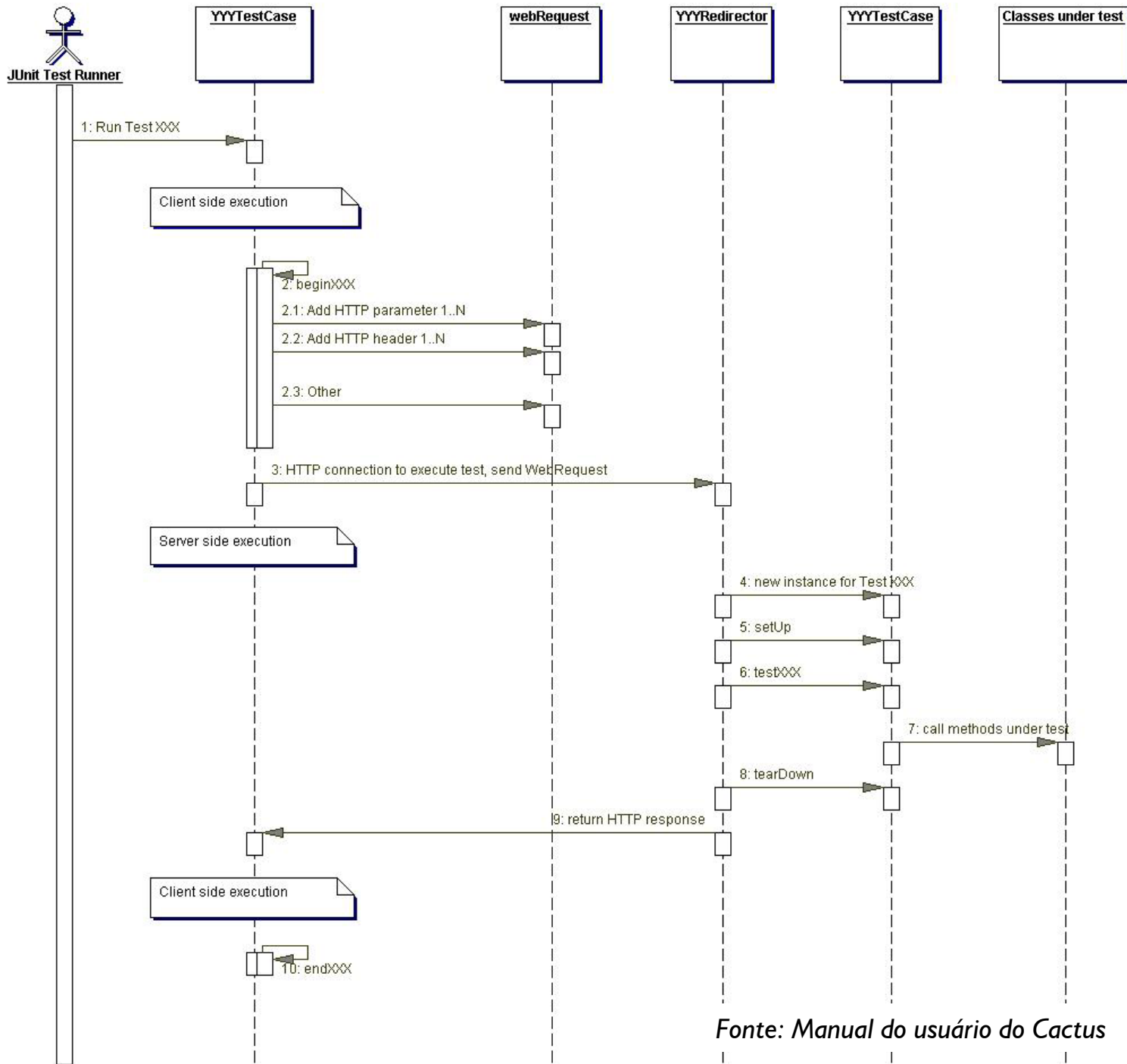


Diagrama UML



- Para cada método `XXX()` a ser testado, pode haver:
 - Um `beginXXX()`, para inicializar a requisição do cliente
 - encapsulada em um objeto `WebRequest` a ser enviado ao servidor
 - Um `testXXX()`, para testar o funcionamento do método no servidor (deve haver ao menos um)
 - Um `endXXX()`, para verificar a resposta do servidor
 - devolvida em um objeto `WebResponse` retornada pelo servidor
- Além desses três métodos, cada `TestCase` pode conter
 - `setUp()`, opcional, para inicializar objetos no servidor
 - `tearDown()`, opcional, para liberar recursos no servidor
- Os métodos do lado do servidor têm acesso aos mesmos objetos implícitos disponíveis em um servlet ou página JSP: `request`, `response`, etc.

- Veja **cactusdemo.zip** (distribuído com esta palestra)
 - Usa duas classes: um servlet (**MapperServlet**) e uma classe (**SessionMapper**) que guarda cada parâmetro como atributo da sessão e em um `HashMap` - veja fontes em **src/xptoolkit/cactus**
- Para rodar, configure o seu ambiente:
 - **build.properties** - localização dos JARs usados pelo servidor Web (`CLASSPATH` do servidor)
 - **runtests.bat** (para Windows) e **runtests.sh** (para Unix) - localização dos JARs usados pelo JUnit (`CLASSPATH` do cliente)
 - **lib/client.properties** (se desejar rodar cliente e servidor em máquinas separadas, troque as ocorrências de `localhost` pelo nome do servidor)
- Para montar, execute:
 - 1. **ant test-deploy** instala `cactus-tests.war` no tomcat
 - 2. o servidor (Tomcat 4.0 startup)
 - 3. **runtests.bat** roda os testes no JUnit

veja demonstração

cactusdemo.zip

- O objetivo deste servlet é
 - 1) gravar qualquer parâmetro que receber na sessão (objeto session)
 - 2) devolver uma página contendo os pares nome/valor em uma tabela
 - 3) imprimir resposta em caixa-alta se `<init-param> ALL_CAPS` definido no `web.xml` contiver o valor `true`

```
public void doGet(...) throws IOException {
    SessionMapper.mapRequestToSession(request);
    writer.println("<html><body><table border='1'>");
    // (... loop for each parameter ...)
    if (useAllCaps()) {
        key = key.toUpperCase();
        val = val.toUpperCase();
    }
    str = "<tr><td><b>"+key+"</b></td><td>"+val+"</td></tr>";
    writer.println(str);
    // (...)
    writer.println("</table></body></html>");
}
```

(1) Grava request em session

(3) Retorna true se `<init-param> "ALL_CAPS"` contiver "true"

(2) Trecho de MapperServlet.java

- Escreveremos os testes para avaliar esses objetivos

MapperServletTest.java

```

public class MapperServletTest extends ServletTestCase { (...)
    private MapperServlet servlet;
    public void beginDoGet(WebRequest cSideReq) {
        cSideReq.addParameter("user", "Jabberwock");
    }
    public void setUp() throws ServletException {
        this.config.setInitParameter("ALL_CAPS", "true");
        servlet = new MapperServlet();
        servlet.init(this.config);
    }
    public void testDoGet() throws IOException {
        servlet.doGet(this.request, this.response);
        String value = (String) session.getAttribute("user");
        assertEquals("Jabberwock", value);
    }
    public void tearDown() { /* ... */ }
    public void endDoGet(WebResponse cSideResponse) {
        String str = cSideResponse.getText();
        assertTrue(str.indexOf("USER</b></td><td>JABBERWOCK") > -1);
    }
}
    
```

Simula DD
<init-param>

Simula servlet container

Verifica se parâmetro foi mapeado à sessão

Verifica se parâmetro aparece na tabela HTML

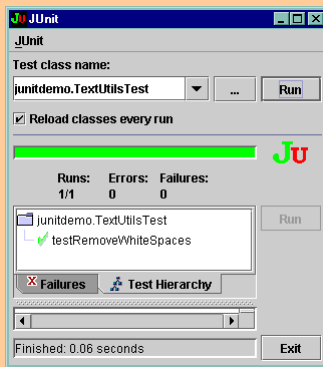
Exemplo: funcionamento

Cliente (JUnit)

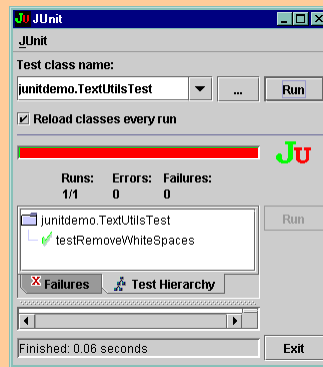
beginDoGet (WebRequest req)

- Grava parâmetro:
nome = **user**
value = **Jabberwock**

SUCCESS!!



FAIL!



endDoGet (WebResponse res)

- Verifica se resposta contém
USER</td><td>**JABBERWOCK**

Servidor (Tomcat)

ReqInfo

setUp ()

- Define init-params
no objeto config
- Roda init (config)

2 conexões HTTP:

- Uma p/ rodar os testes e obter saída do servlet
- Outra para esperar resultados de testes (info sobre exceções)

testDoGet ()

- Roda doGet ()
- Verifica se parâmetro
(no response) foi
mapeado à sessão

TestInfo

tearDown ()

Output

falha local

falha remota

&

- Onde encontrar
 - `http://httpunit.sourceforge.net`
- Framework para testes funcionais de interface (teste tipo "caixa-preta")
 - Verifica a resposta de uma aplicação Web ou página HTML
 - É teste funcional caixa-preta (não é "unit")
 - Oferece métodos para "navegar" na resposta
 - links, tabelas, imagens
 - objetos DOM (Node, Element, Attribute)
- Pode ser combinado com Cactus no `endXXX()`
 - Argumento `com.meterware.httpunit.WebResponse`
- Acompanha **ServletUnit**
 - stub que simula o servlet container

veja também

[httpunitdemo.zip](#)

- **WebConversation**
 - *Representa uma sessão de cliente Web (usa cookies)*

```
WebConversation wc = new WebConversation();  
WebResponse resp = wc.getResponse("http://xyz.com/t.html");
```
- **WebRequest**
 - *Representa uma requisição*
- **WebResponse**
 - *Representa uma resposta. A partir deste objeto pode-se obter objetos **WebLink**, **WebTable** e **WebForm***
- **WebLink**
 - *Possui métodos para extrair dados de links de hipertexto*
- **WebTable**
 - *Possui métodos para navegar na estrutura de tabelas*
- **WebForm**
 - *Possui métodos para analisar a estrutura de formulários*

- Troque o **WebResponse** em cada **endXXX()** por **com.meterware.httpunit.WebResponse**

```
public void endDoGet(com.meterware.httpunit.WebResponse resp)
                    throws org.xml.sax.SAXException {
    WebTable[] tables = resp.getTables();
    assertNotNull(tables);
    assertEquals(tables.length, 1); // só há uma tabela
    WebTable table = tables[0];
    int rows = table.getRowCount();
    boolean keyDefined = false;
    for (int i = 0; i < rows; i++) {
        String key    = table.getCellAsText(i, 0); // col 1
        String value  = table.getCellAsText(i, 1); // col 2
        if (key.equals("USER")) {
            keyDefined = true;
            assertEquals("JABBERWOCK", value);
        }
    }
    if (!keyDefined) {
        fail("No key named USER was found!");
    }
}
```


- **Testes em taglibs (JspRedirector)**
 - Veja exemplos em `cactusdemo/taglib/src`
- **Testes em filtros (FilterRedirector)**
 - Usa proxy `FilterRedirector`
 - Teste básico é verificar se método `doFilter()` foi chamado
 - Veja exemplos em `cactusdemo/src/xptoolkit/AuthFilter`
- **Testes indiretos em páginas JSP (camada **V**iew)**
 - Ideal é JSP não ter código Java
 - Principais testes são sobre a interface: `HttpUnit!`
- **Testes indiretos em EJB (camada **M**odel)**
 - Indireto, através dos redirectors + `JUnitEE`
 - Redirectors permitem testar EJBs com interface local ou remota chamados por código no servidor

veja

[taglibdemo.zip](#)

veja também

[strutsdemo.zip](#)

veja

[hellojsp.zip](#)

veja

[helloejb.zip](#)

Testes em aplicações Web: conclusões

- Aplicações Web são difíceis de testar porque dependem da comunicação com servlet containers
 - Stubs, proxies e APIs, que estendem ou cooperam com o JUnit, tornam o trabalho mais fácil
 - Neste bloco, conhecemos três soluções que facilitam testes de unidade, de integração e de caixa-preta em aplicações Web
- **Stubs** como **ServletUnit** permitem testar as **unidades** de código mesmo que um servidor não esteja presente
- **Proxies** como os "redirectors" do **Cactus** permitem testar a **integração** da aplicação com o container
- Uma **API**, como a fornecida pelo **HttpUnit** ajuda a testar o **funcionamento** da aplicação do ponto de vista do usuário



parte 3



Integração Contínua

com **CVS**, **CruiseControl**,
AntHill e **Gump**



cruisecontrol.sourceforge.net
www.urbanocode.com/projects/anthill
www.cvshome.org
jakarta.apache.org/gump

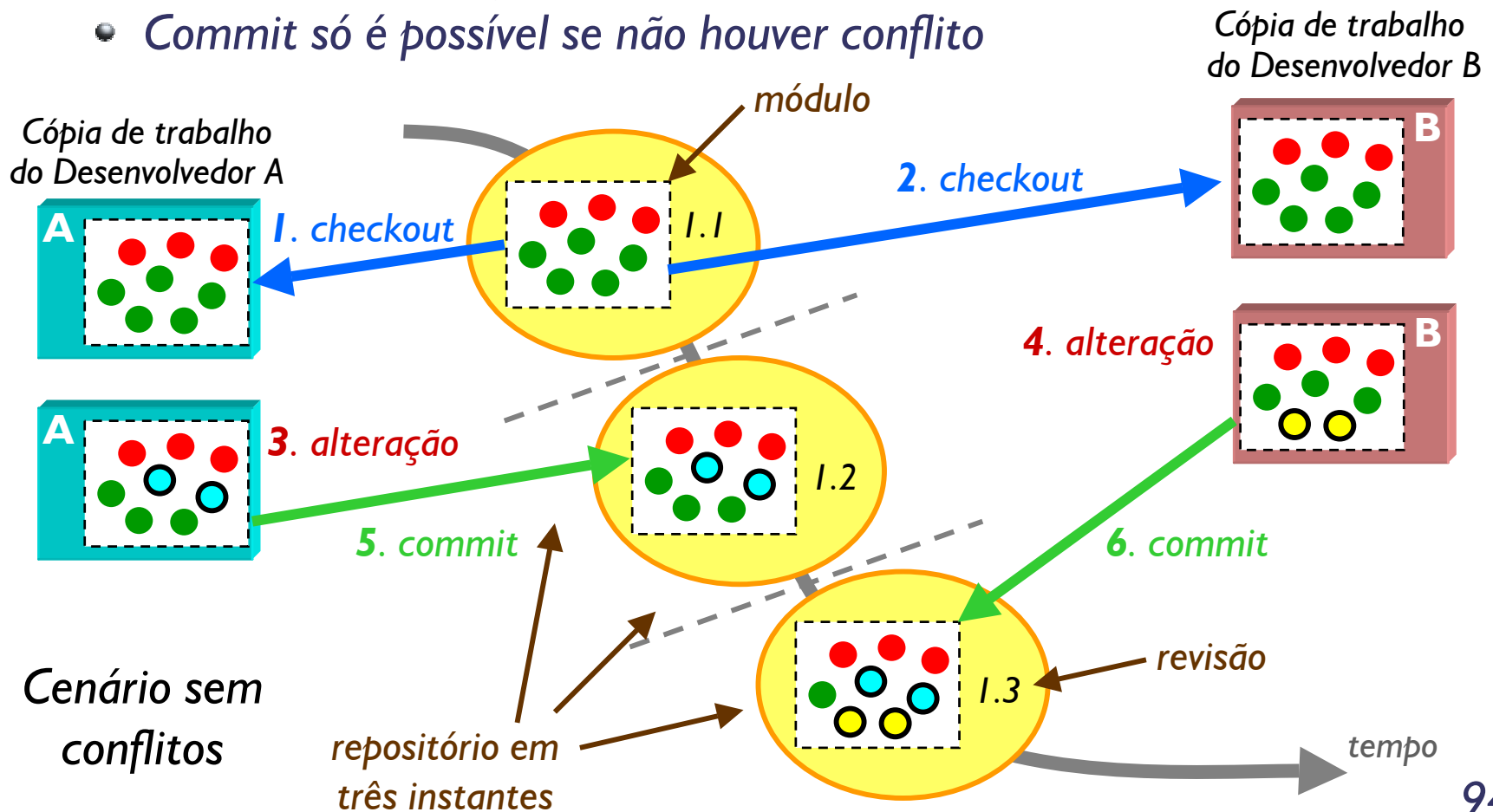


- Um dos requisitos para implementar a integração contínua é ter um sistema onde se possa obter as fontes mais recentes
- Ferramentas **SCM** (Software Configuration Management)
 - **Essenciais** em qualquer projeto sério (grande ou pequeno)
 - No mínimo, mantêm **histórico** do processo de desenvolvimento
 - Existem soluções comerciais e open-source: CVS, Perforce, ClearCase, SourceSafe, StarTeam, Merant PVCS, Continuum, etc.
- Esta seção apresentará um breve tutorial da mais popular ferramenta SCM open-source: **CVS**
- Serão apresentadas três ferramentas de integração contínua, que combinam o Ant, JUnit e um SCM
 - ThoughtWorks **CruiseControl** (suporta Ant, JUnit, vários SCM)
 - UrbanCode **AntHill** (suporta Ant, JUnit, CVS*)
 - Jakarta **Gump** (suporta Ant, JUnit, CVS)

* Oferece interface para suporte de outros SCM

- **C**oncurrent **V**ersions **S**ystem
 - Sistema de controle de versões open-source
 - Baseado em um repositório central onde usuários podem fazer atualizações (commit) e downloads (checkout)
- **R**epositório **C**VS
 - Mantém uma cópia de todos os arquivos e diretórios sob controle de versões (usa formato **RCS*** - arquivo com extensão ".v" guarda todo histórico de modificações)
 - Permite a recuperação de quaisquer versões anteriormente armazenadas de arquivos texto
- **D**iretório de trabalho (cópia de trabalho)
 - Criado por cada desenvolvedor
 - Contém cópia local dos arquivos baixados do repositório através de uma operação de **checkout**
 - Cada pasta e subpasta contém uma pasta especial "**CVS**"

- **Desenvolvedores baixam última versão do repositório**
 - *Trabalham em cópia local de módulo. Ao terminar, fazem upload (commit) e alterações são mescladas em nova revisão*
 - *Commit só é possível se não houver conflito*

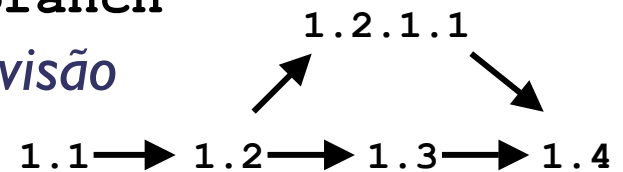


CVS: repositório e operações

- Repositório CVS é acessado por um cliente que precisa saber
 - Caminho ou endereço da raiz do repositório
 - Método de acesso que será utilizado
 - Dados para autenticação no servidor
- Essas informações podem ser guardadas em uma variável de ambiente do sistema CVSROOT

```
set CVSROOT=:local:/usr/cvs/root
set CVSROOT=:pserver:helder:pz@192.168.1.1:/usr/cvs/root
```
- Uma vez definido o CVSROOT, pode-se
 - criar uma cópia de trabalho de um módulo do repositório :
> `cvs checkout hello cvs`
 - sincronizar a cópia local com o repositório:
> `cd hello cvs`
> `cvs update`
 - gravar alterações feitas localmente no repositório
> `cvs commit -m "Novo método sayBye() em Hello.java"`

- **Número de revisão (controle por arquivo)**
 - Número gerado pelo sistema para identificar cada modificação feita em um arquivo
 - Começa em 1.1.1.1, seguido por 1.2. Depois tem o segundo número incrementado a cada commit
- **Tag (controle por módulo)**
 - Usado para rotular uma versão do módulo com um nome
 - Comando: `cv`s tag nome_do_release
- **Branch (abre sub-projeto)**
 - Comando: `cv`s tag -b nome_do_branch
 - Usam dígitos extras no número de revisão
 - Podem depois ser incorporados ao projeto principal: `cv`s update -r nome_do_branch



Desenvolvimento típico com CVS

SUCESU-SP 2002

A. Inicialização do ambiente

- Fazer **checkout** de módulo em diretório de trabalho

B. Um ciclo de desenvolvimento (geralmente curto)

- Fazer alterações em código-fonte, criar novos arquivos e adicioná-los ao CVS, remover arquivos
- Compilar, montar localmente e rodar testes
- Fazer **update** para incorporar eventuais mudanças feitas por outros desenvolvedores
- Resolver eventuais conflitos
- Compilar, montar e rodar testes de novo
- Cometer todas as mudanças: **commit**

C. Lançamento: após vários ciclos

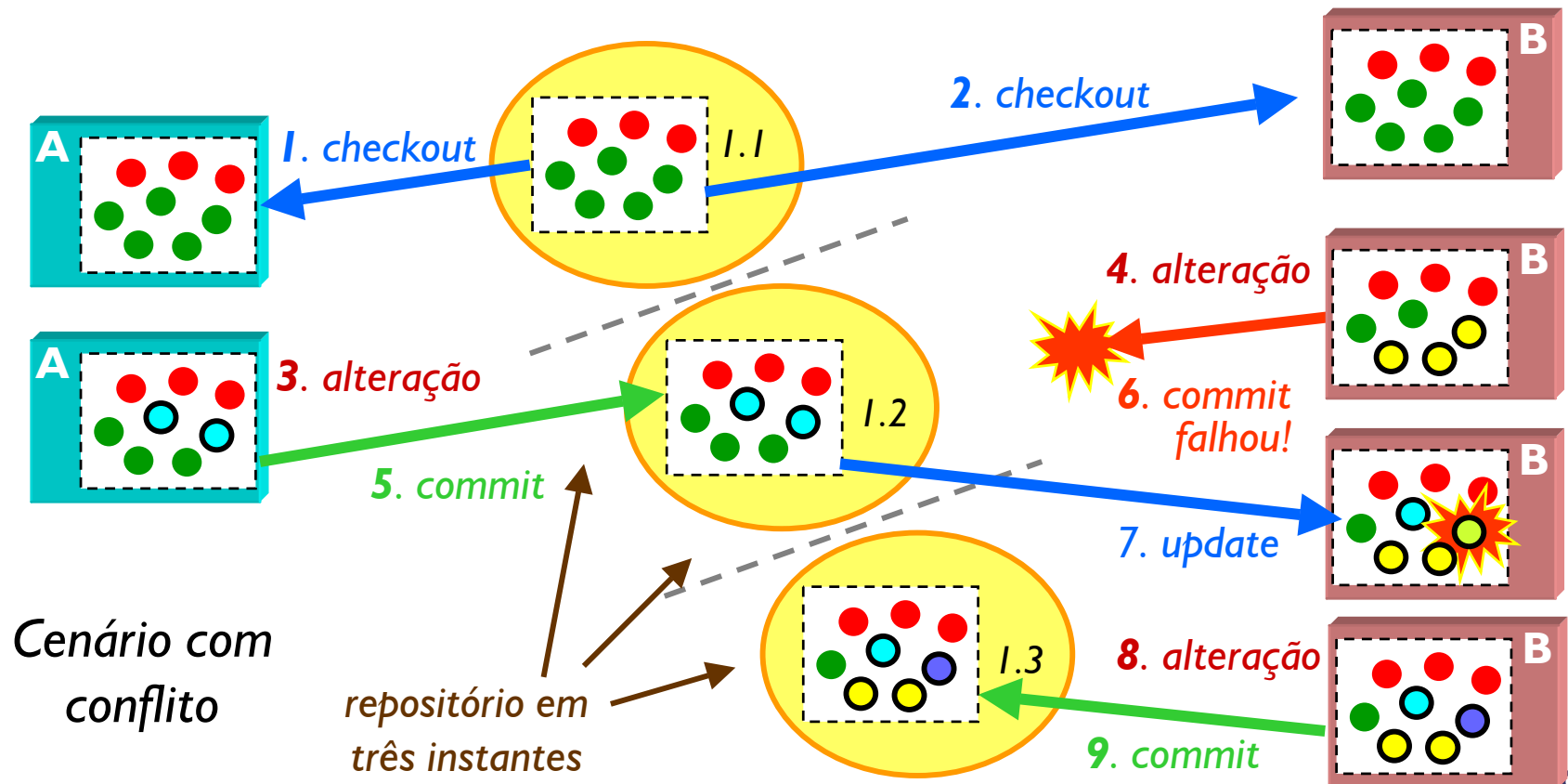
- Rotular o lançamento com um **tag**

Adição e remoção de arquivos

SUCESU-SP 2002

- Principais comandos usados para adicionar e remover arquivos do repositório
 - > **cv**s **add** nome_do_arquivo
 - > **cv**s **remove** nome_do_arquivo
 - Adiciona ou remove um novo arquivo ao repositório (a mudança só será efetivada após um commit)
 - > **cv**s **import** contexto/modulo *identif start*
 - Importa uma árvore de arquivos e diretórios para o repositório
 - Semelhante a checkout mas não cria diretórios CVS (ou seja, não cria cópia de trabalho, apenas versão para distribuição)
- Arquivos binários
 - É preciso rotular arquivos binários com um flag especial antes de importar arquivos ou adicioná-lo ao repositório
 - > **cv**s **add** **-kb** imagem.gif
 - Se isto não for feito, **dados serão perdidos!**

- Ocorrem quando dois usuários alteram mesma área do código
 - Primeiro que fizer commit grava as alterações
 - Outro usuário só pode cometer suas mudanças depois que atualizar sua cópia de trabalho e resolver o conflito



- *Revisão 1.27 do repositório contém:*

```
public class HelloWorld {  
    public String sayHello() {  
        return "Hello world...";  
    }  
}
```

- *Cópia de trabalho (não sincronizada) contém:*

```
public class HelloWorld {  
    public String sayHello() {  
        return "Hello, world!";  
    }  
}
```

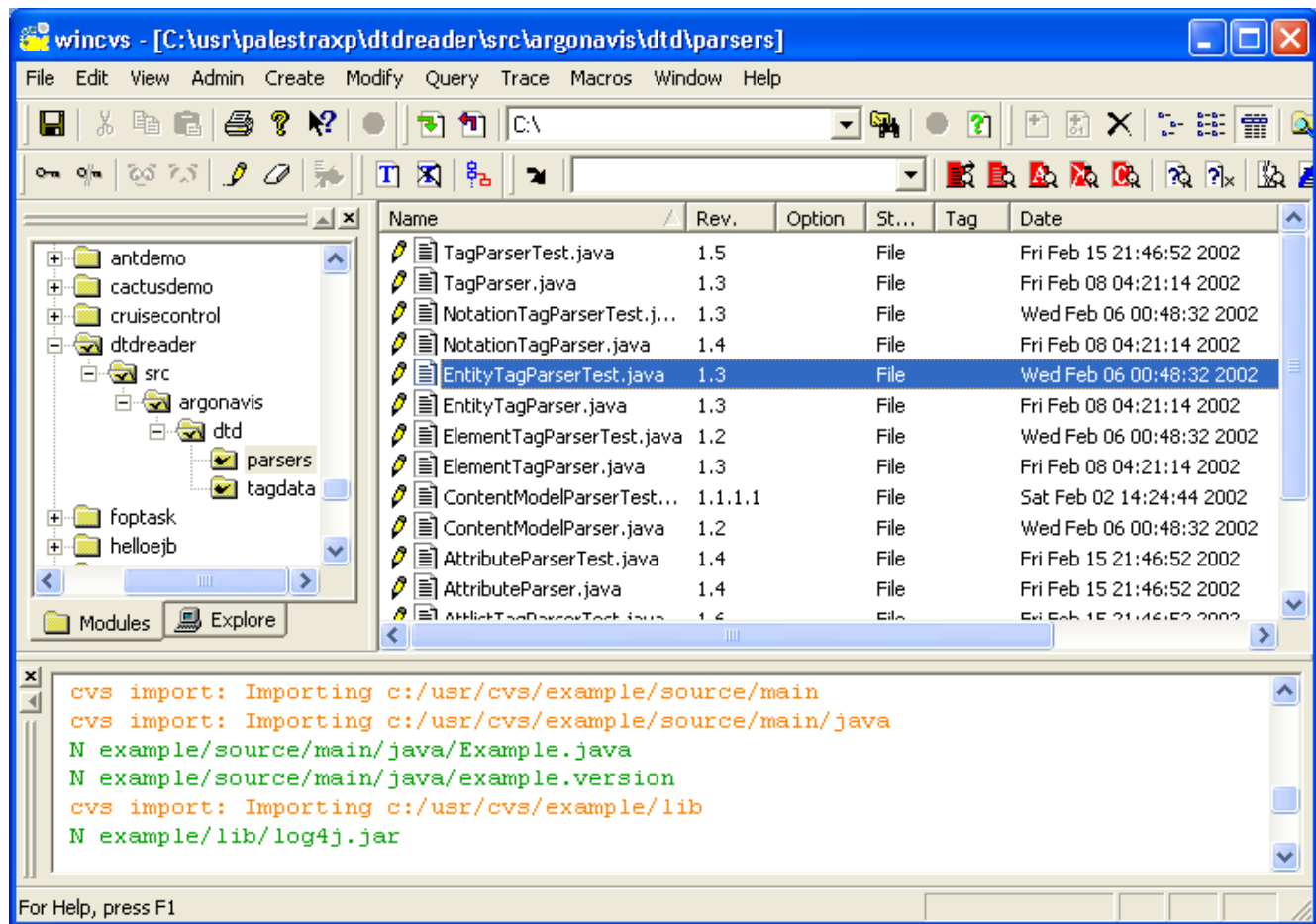
- *Depois do **update**, cópia de trabalho contém **merge**:*

```
public class HelloWorld {  
    public String sayHello() {  
<<<<<< HelloWorld.java  
        return "Hello, world!";  
=====  
        return "Hello world...";  
>>>>>> 1.27  
    }  
}
```

É preciso fazer as alterações e
remover os <<< == >>> antes
de tentar novo commit

Cliente gráfico: WinCVS

- Interface gráfica para uso de CVS em ambiente Windows
- Projeto open-source: www.cvsogui.org



- *Ant suporta CVS através do elemento **< cvs >***
 - *Ant também suporta outros sistemas de controle de versões*
 - *Deve haver um cliente CVS acessível por linha de comando*
- *Exemplos*

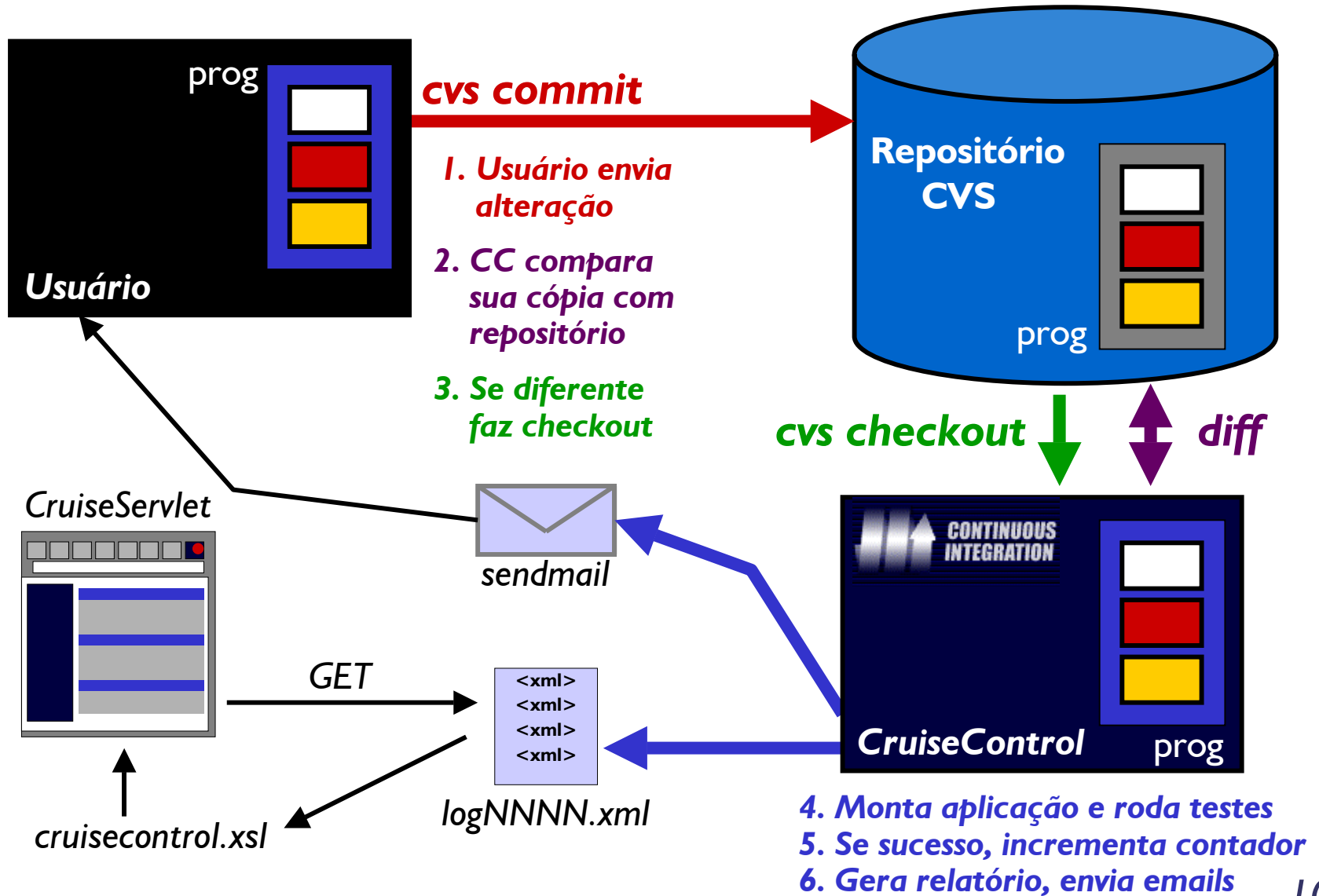
```
<target name="cvs-commit" depends="build" >  
  <cvs command="commit" />  
  <cvs command="tag ${cvs.release.tag}" />  
</target>
```

```
<target name="cvs-export" depends="build" >  
  <cvs command="export -d ${export.base} -r  
    ${cvs.release.tag}  
    ${project.name}"  
    dest="${basedir}/.." />  
</target>
```

- *Ferramenta para integração contínua e automática*
 - *Ideal para integrar software desenvolvido em equipe*
 - *Baseada na ferramenta Ant, através da qual opera sistema de controle de revisões (CVS, ClearCase, Perforce, StarTeam, ou outro)*
 - *Artigo "Continuous Integration" [8] (Fowler / Foemmel)*
- *Roda em um servidor onde periodicamente...*
 - *... monta toda a aplicação*
 - *... roda todos os testes*
 - *... gera relatórios sobre os resultados em XML (enviados por e-mail para os "committers" e publicados em página na Web)*
- *Viabiliza prática de "lançamentos pequenos" do XP*
 - *Repositório sempre contém a última versão estável*

CruiseControl: funcionamento

SUCESU-SP 2002



Relatórios gerados a cada build

CONTINUOUS INTEGRATION

BUILD FAILED

Ant Error Message: C:\us\palestraxp\cruisecontrol\bellatrix\cruisedemo\build.xml:81: Test test.hello.HelloWorldTe

Date of build: July 26 2002

Time to build: 40 seconds

Last changed: 2002-Jul-26 15:41:33

Last log entry: Consertei o metodo upper.

Next Build Starts At:
07/26/2002 15:46

Previous Builds:

- 26/07/2002 15h44min0s BRT (Teste.8)
- 26/07/2002 15h41min0s BRT
- 26/07/2002 1h21min0s BRT (Teste.7)
- 26/07/2002 1h16min0s BRT
- 26/07/2002 1h14min0s BRT
- 25/07/2002 11h34min0s BRT
- 25/07/2002 11h28min0s BRT (Teste.6)
- 25/07/2002 11h1min0s BRT
- 25/07/2002 10h54min0s BRT (Teste.5)
- 25/07/2002 10h52min0s BRT
- 25/07/2002 10h49min0s BRT

Unit Tests: (2)

failure	testUpper
---------	-----------

Unit Test Error Details: (1)

Test: testUpper

Type: junit.framework.AssertionFailedError

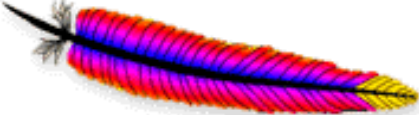
Message: expected:<TEXT IN UPPERCASE> but was:<text in uppercase>

```
junit.framework.AssertionFailedError: expected:<TEXT IN UPPERCASE> but was:<text in uppercase>
    at junit.framework.Assert.fail(Assert.java:51)
    at junit.framework.Assert.failNotEquals(Assert.java:234)
    at junit.framework.Assert.assertEquals(Assert.java:68)
    at junit.framework.Assert.assertEquals(Assert.java:75)
    at test.hello.HelloWorldTest.testUpper(HelloWorldTest.java:25)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)
    at java.lang.reflect.Method.invoke(Method.java:324)
    at junit.framework.TestCase.runTest(TestCase.java:166)
    at junit.framework.TestCase.runBare(TestCase.java:140)
    at junit.framework.TestResult$1.protect(TestResult.java:106)
    at junit.framework.TestResult.runProtected(TestResult.java:124)
    at junit.framework.TestResult.run(TestResult.java:109)
    at junit.framework.TestCase.run(TestCase.java:131)
    at junit.framework.TestSuite.runTest(TestSuite.java:173)
    at junit.framework.TestSuite.run(TestSuite.java:168)
```

veja demonstração

cruisedemo.zip

- **Ferramenta para integração contínua**
 - *Aplicação Web que roda builds agendados*
 - *Suporte SCM é limitado a CVS (usuário pode escrever driver para outro SCM se desejar)*
 - *Publica artefatos: código-fonte, documentação, links para distribuição, saída do build*
- **Vantagens sobre CruiseControl**
 - *Tudo é feito via interface Web*
 - *Configuração é muito mais simples*
 - *Suporta múltiplos projetos*
 - *Não requer alterações no build.xml*
 - *Rotula versões (tag) automaticamente*
- **Desvantagens**
 - *Suporta apenas CVS*
 - *Não gera automaticamente relatórios de testes JUnit*



- *Ferramenta de integração contínua do Projeto Jakarta*
 - *Realiza a integração não só da aplicação mas, opcionalmente, de todas as suas dependências (compila e monta cada uma delas)*
 - *Fornece acesso a JavaDocs, referências cruzadas (interdependências) e JARs dos projetos montados*
 - *Instalação e configuração não são triviais*
- *Gump separa as configurações em arquivos XML distintos que podem ser reutilizados*
 - **project** *Define os JARs que um projeto exporta (que servirão de dependências para outros projetos)*
 - **module** *Coleção de projetos guardados em um repositório*
 - **repository** *Informação sobre como acessar os repositórios CVS*
 - **profile** *Coleção de projetos e repositórios*
 - **workspace** *Configurações globais, sistema, etc.*

Gump: funcionamento

Task Name	Time	Component	Status
excilbur-util	03:52:23	commons-fileupload	SUCCESS
excilbur-xmlutil	03:52:26	excilbur-concurrent	SUCCESS
excilbur-zip	03:52:29	excilbur-extension	SUCCESS
eyebrowse	03:52:32	excilbur-util	SUCCESS
fop-buildtools	03:52:36	jakarta-cactus-framework-13	SUCCESS
Forrest	03:52:51	jakarta-lucene	SUCCESS
freemarker	03:53:18	jaxen	SUCCESS
gump	03:53:34	xml-soap	SUCCESS
hsqldb	03:54:05	commons-dbc	SUCCESS
httpunit	03:54:11	commons-messenger	SUCCESS
jakarta-alexandria	03:54:16	commons-modeler	SUCCESS
jakarta-ant	03:54:21	dist-ant	SUCCESS
jakarta-ant-antidote	03:57:05	dist-bsf	SUCCESS
jakarta-ant-myrmidon	03:57:16	excilbur-altmi	SUCCESS
jakarta-avalon	03:57:24	excilbur-instrument	FAILED
jakarta-avalon-cornerstone	03:57:29	excilbur-pool	PREREQ FAILURE - excilbur-instrument
jakarta-avalon-dist	03:57:29	excilbur-component	PREREQ FAILURE - excilbur-pool
jakarta-avalon-excalibur	03:57:29	excilbur-thread	PREREQ FAILURE - excilbur-instrument
jakarta-avalon-exceptionutil	03:57:29	excilbur-sourcesolve	PREREQ FAILURE - excilbur-pool
jakarta-avalon-ftpserver	03:57:29	excilbur-configuration	SUCCESS
jakarta-avalon-logkit			
jakarta-avalon-phoenix			
jakarta-bcel			
jakarta-cactus-ant			
jakarta-cactus-distribution-12			
jakarta-cactus-distribution-13			
jakarta-cactus-distribution-src			
jakarta-cactus-documentation			
jakarta-cactus-framework-12			
jakarta-cactus-framework-13			
jakarta-cactus-sample-servlet-12			
jakarta-cactus-sample-servlet-13			
jakarta-ecs			

- Duas etapas
 - Geração de scripts
 - Execução dos scripts
- Geração cria scripts usando configuração do workspace
- Execução (que pode ser automatizada) usa os outros arquivos para montar as dependências e gerar relatórios
- Relatórios, acessíveis via Web mostram conflitos
 - Pode enviar e-mail em caso de falha

Conclusões: ferramentas de integração contínua

SUCESU-SP 2002

- A tabela abaixo apresenta uma breve comparação entre as ferramentas de integração contínua analisadas

Recurso	CruiseControl	AntHill	Gump
Instalação e configuração	Média dificuldade	Fácil	Difícil
Requer alterações em buildfiles	Sim	Não	Não
Monta dependências	Não	Não	Sim
Controla SCM automaticamente	Não. Comandos têm que ser incluídos no buildfile	Sim	Sim
SCMs suportados	CVS, VSS, ClearCase, MKS, Perforce, PVCS, StarTeam	CVS	CVS
Suporte a múltiplos projetos simultâneos	Requer nova instância da aplicação executando e outro buildservlet.war	Requer adição de nova definição na interface Web	Requer a configuração de arquivos XML



Conclusões



- Neste tutorial, você conheceu
 - **JUnit** - framework criado para facilitar a criação e execução de testes para medir a qualidade do seu software
 - Extensões do JUnit para situações onde testar é difícil
 - **Ant** - ferramenta indispensável que ajuda a automatizar diversos processos comuns em ambientes de desenvolvimento em Java
 - **Cactus** - coleção de redirecionadores para facilitar testes de integração de aplicações Web
 - **CVS** - um popular sistema de controle de versões open-source
 - Ferramentas para automatizar a integração contínua

- *As ferramentas apresentadas neste tutorial podem*
 - *melhorar a qualidade do seu software*
 - *aumentar o reuso de seus componentes*
 - *melhorar suas estimativas de prazos*
 - *melhorar a produtividade de sua equipe*
 - *melhorar a comunicação*
 - *reduzir custos*
 - *tornar o desenvolvimento mais ágil e eficiente*
 - *reduzir drasticamente o tempo gasto na depuração*
- *O único investimento necessário para obter os benefícios é **aprender a usá-las***

- [1] *Richard Hightower e Nicholas Lesiecki. Java Tools for eXtreme Programming. Wiley, 2002. Explora as ferramentas Ant, JUnit, Cactus, JUnitPerf, JMeter, HttpUnit usando estudo de caso com processo XP.*
- [2] *Jeffries, Anderson, Hendrickson. eXtreme Programming Installed, Addison-Wesley, 2001. Contém exemplos de estratégias para testes.*
- [3] *Apache Ant User's Manual. Ótima documentação repleta de exemplos.*
- [4] *Apache Cactus User's Manual. Contém tutorial para instalação passo-a-passo.*
- [5] *Steve Lougran. Ant In Anger - Using Apache Ant in a Production Development System. (Ant docs) Ótimo artigo com boas dicas para organizar um projeto mantido com Ant.*
- [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. Dicas do que fazer ou não fazer para construir bons testes.*

- [8] Martin Fowler, Matthew Foemmel. *Continuous Integration*.
<http://www.martinfowler.com/articles/continuousIntegration.html>. Ótimo artigo sobre integração contínua e o CruiseControl.
- [9] 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.
- [10] Mackinnon, Freeman, Craig. *Endo-testing with Mock Objects*.
<http://mockobjects.sourceforge.net/misc/mockobjects.pdf>. O autor apresenta técnicas para testes usando uma variação da técnica de stubs chamada de "mock objects".
- [11] William Wake. *Test/Code Cycle in XP. Part I: 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.
- [12] Steve Freeman, *Developing JDBC Applications Test First*. 2001. Tutorial sobre metodologia test-first com mock objects para JDBC.
- [13] Martin Fowler. *Refactoring: improving the design of existing code*. Addison-Wesley, 2000. Cap 4 (building tests) é um tutorial usando JUnit.

- [14] Erik Hatcher. *Java Development with Ant*. Manning Publications. August 2002. *Explora os recursos básicos e avançados do Ant, sua integração com JUnit e uso com ferramentas de integração contínua como AntHill e CruiseControl.*
- [15] Jesse Tilly e Erik Burke. *Ant: The Definitive Guide*. O'Reilly and Associates. May 2002. *Contém referência completa e ótimo tutorial sobre recursos avançados como controle dos eventos do Ant e criação de novas tarefas.*
- [16] Per Cederqvist et al. *Version Management with CVS*.
<http://www.cvshome.org/docs/manual/>. *O manual oficial do CVS com ótimo tutorial e referência completa de todos os comandos.*
- [17] Karl Fogel. *Open Source Development with CVS*. Coriolis Press.
<http://cvsbook.red-bean.com/>. *No site o autor disponibiliza parte do livro que contém todos os capítulos sobre como criar, administrar e usar um repositório CVS.*
- [18] *Apache JMeter User's Manual*. *Fonte dos exemplos.*
- [19] Mike Clark, *JUnitPerf Docs*. *Fonte dos exemplos.*

helder@argonavis.com.br

Selecione o link relativo a esta palestra no endereço

www.argonavis.com.br/comdex2002

Recursos disponíveis no site:

- *Palestra completa em PDF*
- *Todo o código-fonte usado nos exemplos e demonstrações*
- *Instruções detalhadas sobre como rodar e instalar os exemplos*
- *Links para software utilizado e documentação*