12

J820

Integração Contínua

com CVS, CruiseControl, AntHill e Gump

Helder da Rocha (helder@acm.org)

argonavis.com.br

Integração contínua

- 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, Continuus, 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

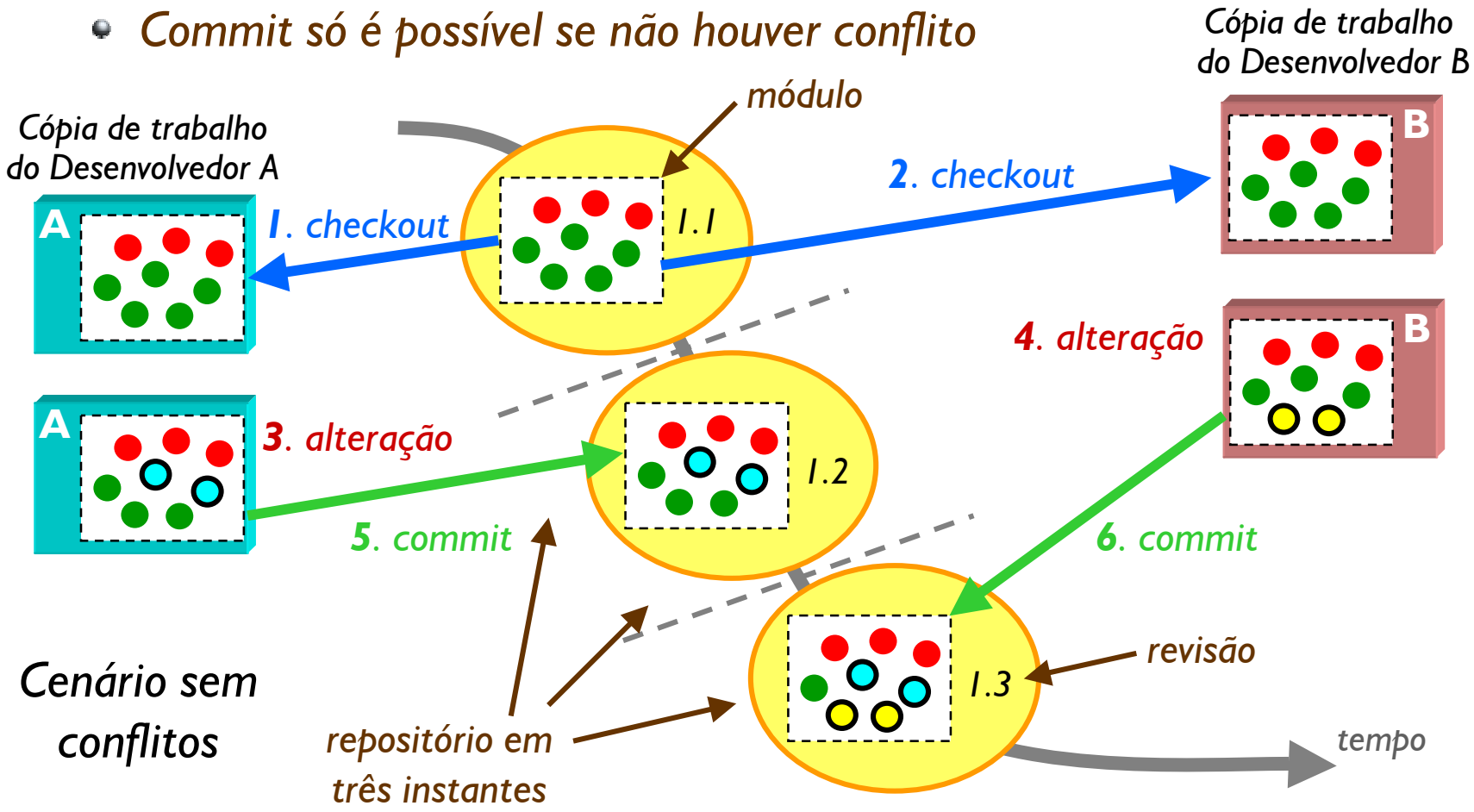
CVS: o que é?

- **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)
- Repositório CVS
  - 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
- Diretório 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"

* Revision Control System

CVS: como funciona

- 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 hellocvs
```

  - sincronizar a cópia local com o repositório:

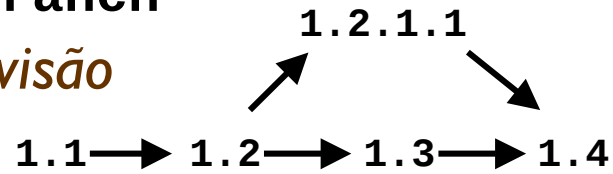
```
> cd hellocvs
> cvs update
```

  - gravar alterações feitas localmente no repositório

```
> cvs commit -m "Novo método sayBye() em Hello.java"
```

Revisão, tag e branch

- 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: `cvs tag nome_do_release`
- Branch (abre sub-projeto)
  - Comando: `cvs tag -b nome_do_branch`
  - Usam dígitos extras no número de revisão
  - Podem depois ser incorporados ao projeto principal: `cvs update -r nome_do_branch`



Desenvolvimento típico com CVS

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

- *Principais comandos usados para adicionar e remover arquivos do repositório*

```
> cvs add nome_do_arquivo
> cvs remove nome_do_arquivo
```

  - *Adiciona ou remove um novo arquivo ao repositório (a mudança só será efetivada após um commit)*

```
> cvs import contexto/modulo identif start
```

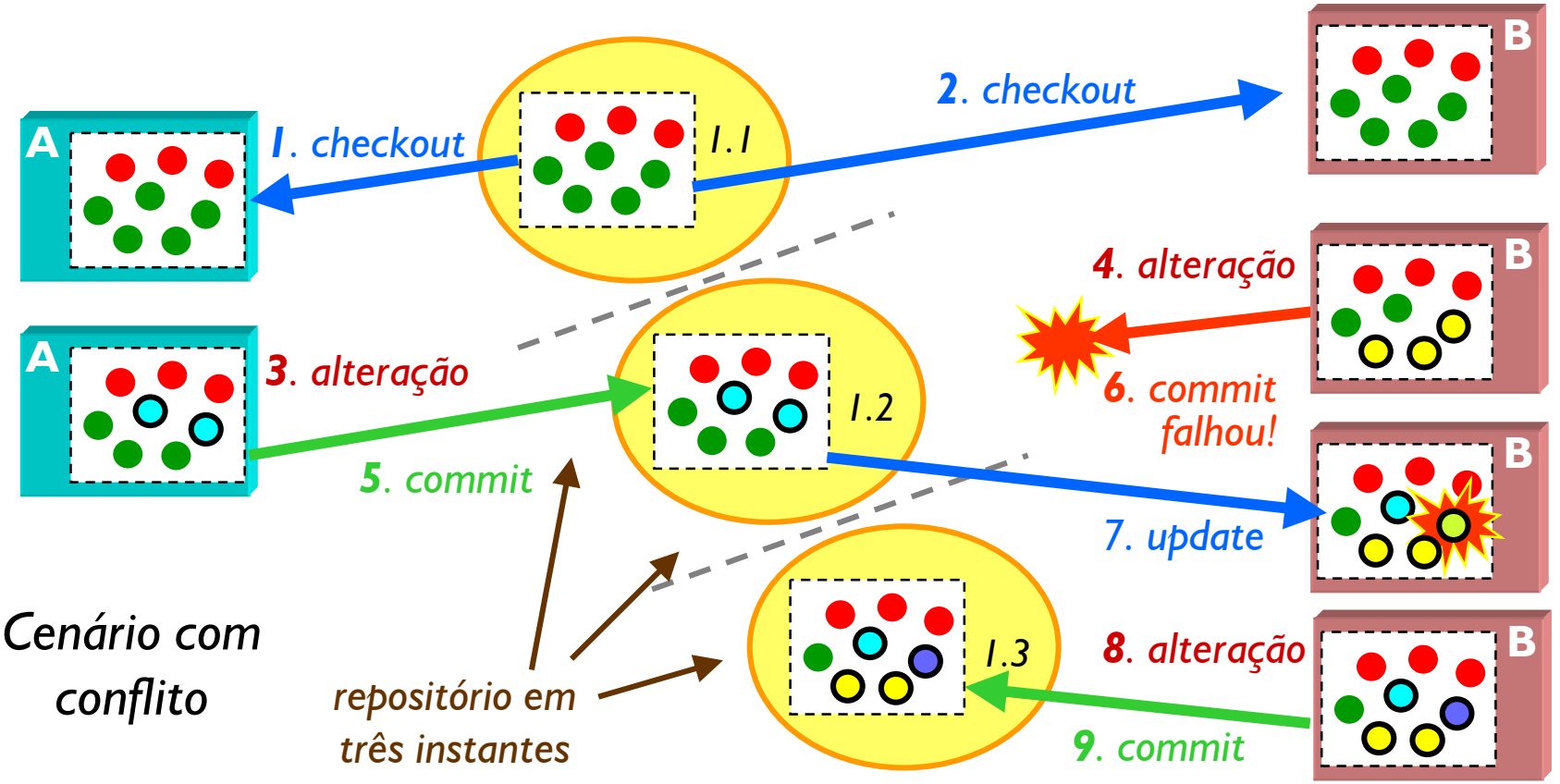
  - *Importa uma árvore de arquivos e diretórios para o repositório*
  - *Semelhante a checkout mas não cria diretórios CVS (ou seja, não cria cópia de trabalho, apenas versão para distribuição)*
- *Arquivos binários*
  - *É preciso rotular arquivos binários com um flag especial antes de importar arquivos ou adicioná-lo ao repositório*

```
> cvs add -kb imagem.gif
```

  - *Se isto não for feito,* ***dados serão perdidos!***

Conflitos

- 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



Resolução de conflitos

- *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!";
   }
}
```

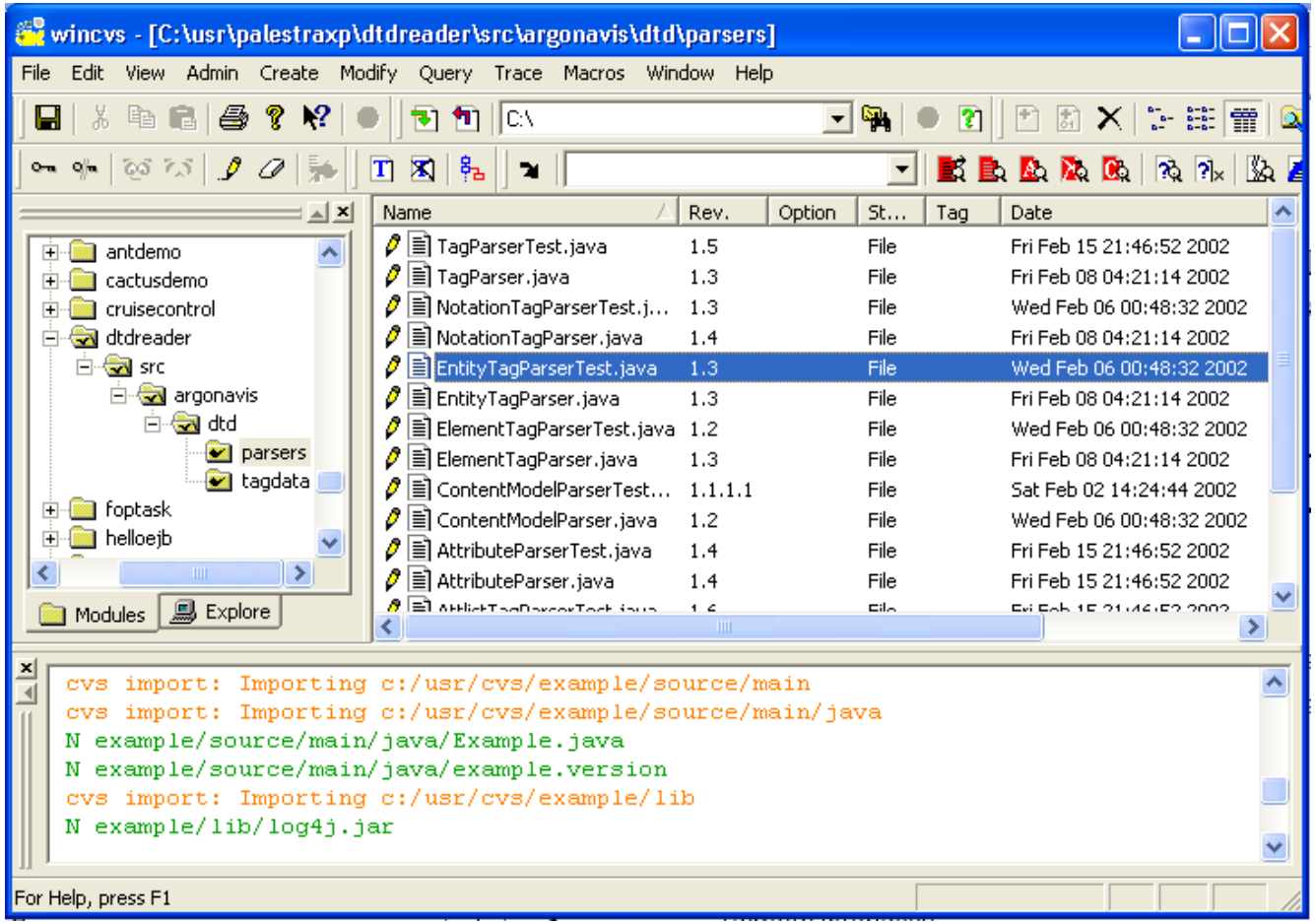
- *Depois do update, cópia de trabalho contém merge:*

```
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.cvsgui.org*



CVS com Ant

- 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

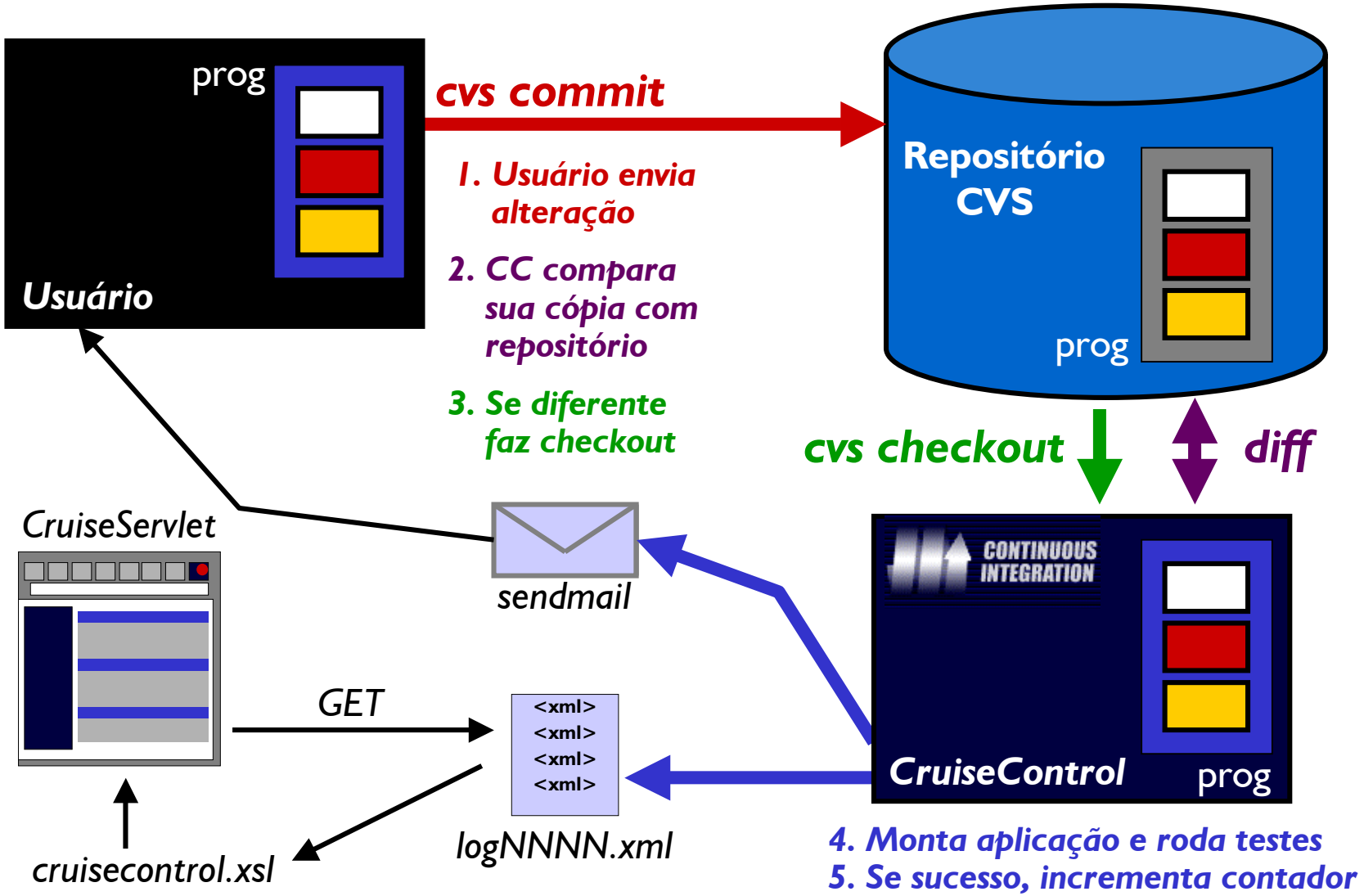
```
<target name="cvs-commit" depends="build" >
    <cvs command="commit" />
    <cvs command="tag ${cvs.release.tag}" />
</target>

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

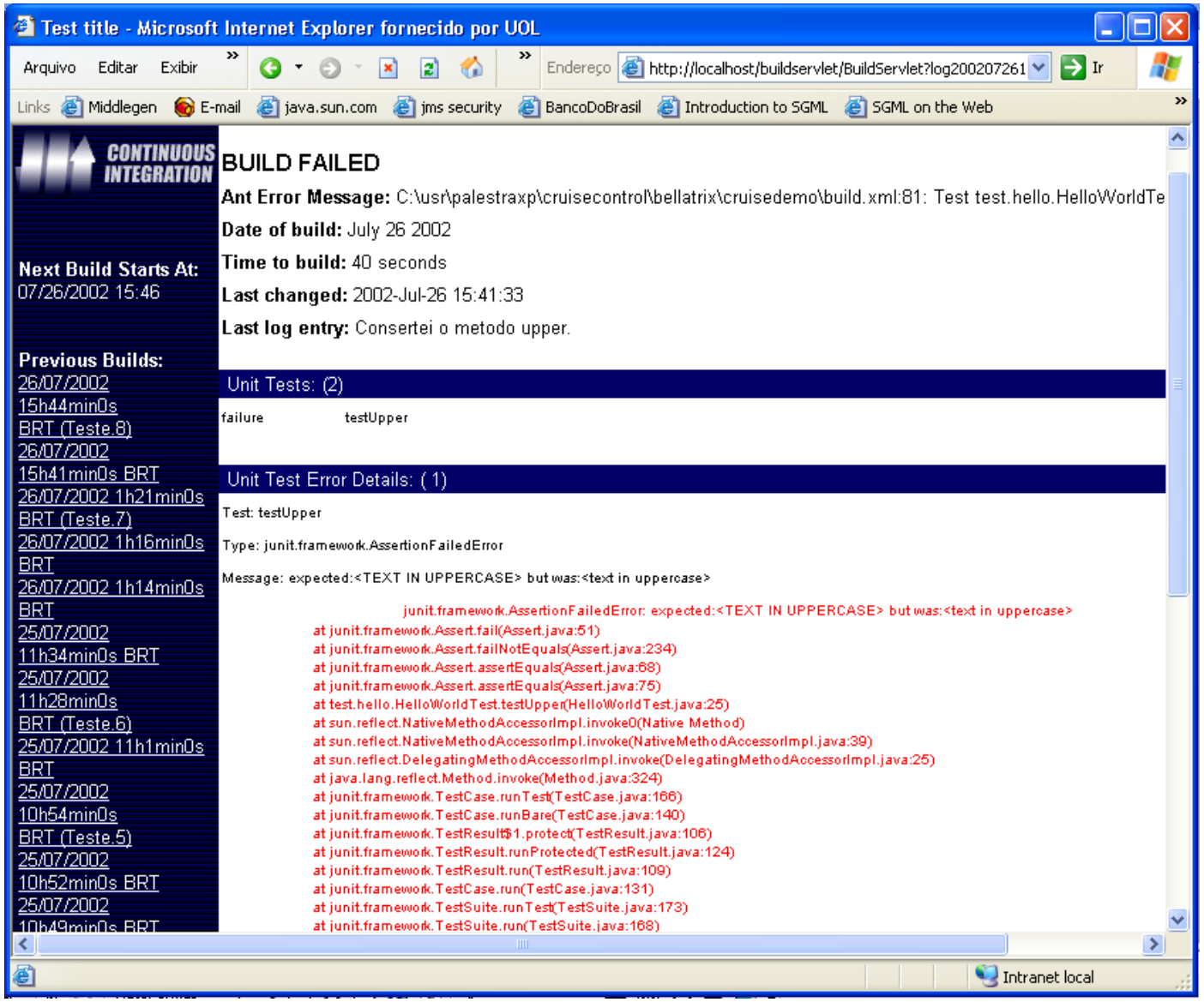
CruiseControl

- 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



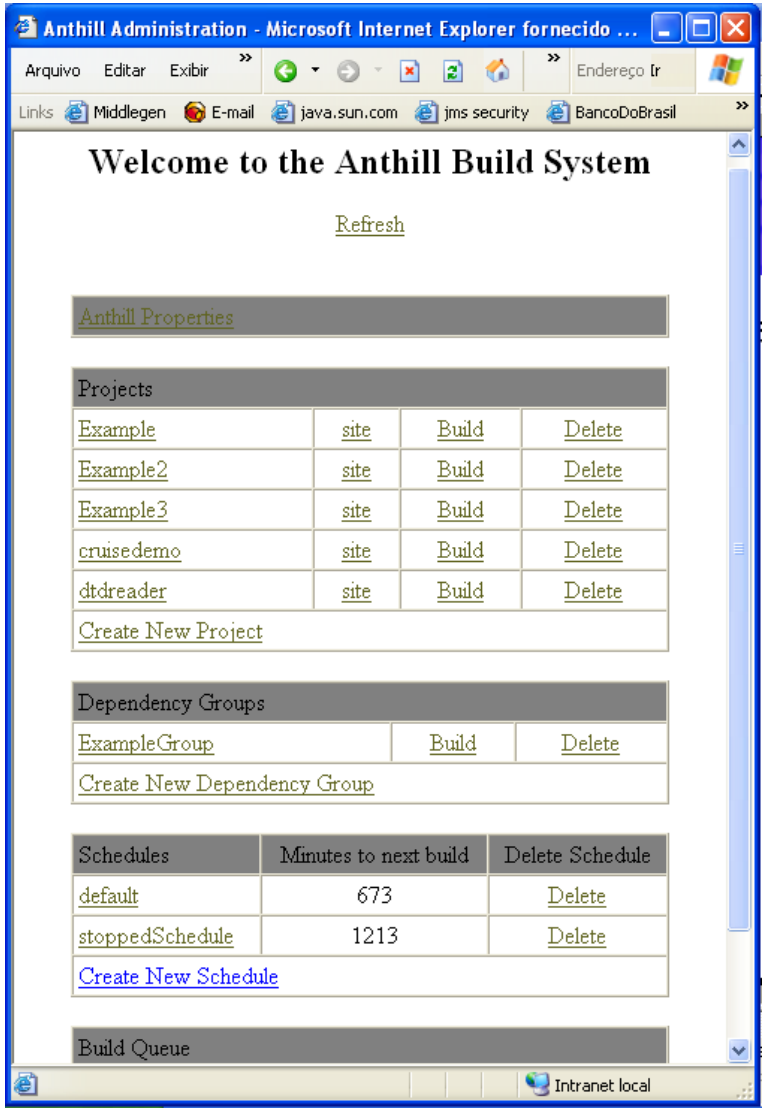
Relatórios gerados a cada build

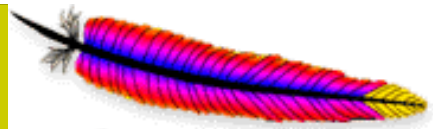


- *Ferramenta para integração contínua*
  - *Aplicação Web que roda builds agendados*
  - *Suporte SCM é limitado a CVS (usuário pode escrever driver para outro SCM se desejar)*
  - *Publica artefatos: código-fonte, documentação, links para distribuição, saída do build*
- *Vantagens sobre CruiseControl*
  - *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*

AntHill: funcionamento

- Cada projeto está associado a um agendamento
- Na hora de realizar o build, AntHill consulta o repositório. Se houver novos arquivos, build ocorre
- Em caso de sucesso
  - Arquivo com número da versão é incrementado
- Em caso de falha
  - Participantes do projeto são notificados por email
- Relatórios são gerados em ambos os casos

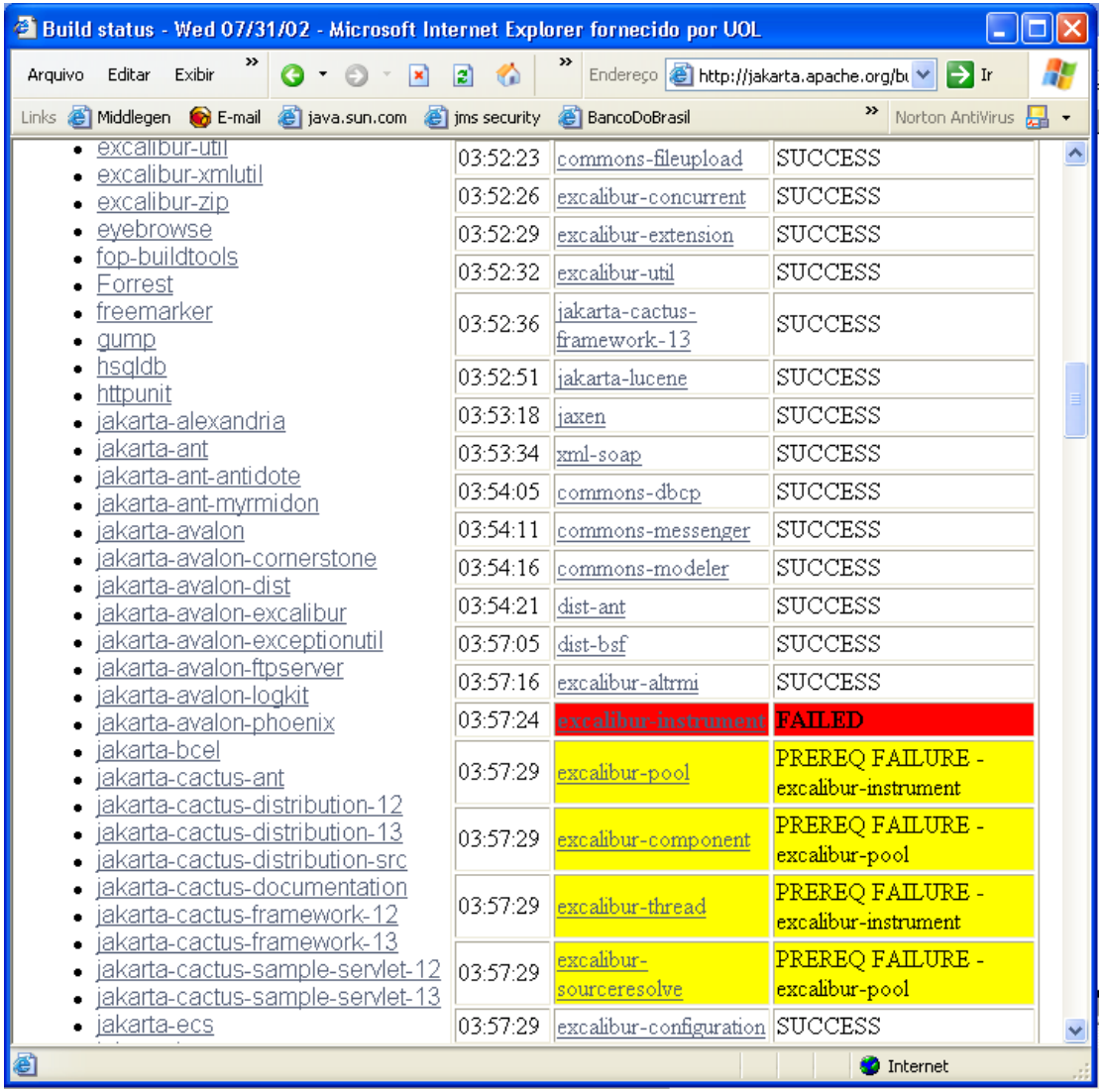




Gump

- *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 (inter-dependências) 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 guardaros em um repositório*
  - *repository* — *Informação sobre como acessar os repositórios CVS*
  - *profile* — *Coleção de projetos e repositórios*
  - *workspace* — *Configurações globais, sistema, etc.*

Gump: funcionamento



excalibur-util	03:52:23	commons-fileupload	SUCCESS
excalibur-xmlutil	03:52:26	excalibur-concurrent	SUCCESS
excalibur-zip	03:52:29	excalibur-extension	SUCCESS
eyebrowse	03:52:32	excalibur-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-dbcp	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	excalibur-altrmi	SUCCESS
jakarta-avalon	03:57:24	excalibur-instrument	FAILED
jakarta-avalon-cornerstone	03:57:29	excalibur-pool	PREREQ FAILURE - excalibur-instrument
jakarta-avalon-dist	03:57:29	excalibur-component	PREREQ FAILURE - excalibur-pool
jakarta-avalon-excalibur	03:57:29	excalibur-thread	PREREQ FAILURE - excalibur-instrument
jakarta-avalon-exceptionutil	03:57:29	excalibur-sourceresolve	PREREQ FAILURE - excalibur-pool
jakarta-avalon-ftpserver	03:57:29	excalibur-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

- 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 incluidos 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 configu-ração de arquivos XML

Fonte: Erik Hatcher, "Java Development with Ant" [14]

Fontes

[1] *Documentação do CruiseControl*. *cruisecontrol.sourceforge.net*

[2] *Erik Hatcher*. *Java Development with Ant*. *Manning, 2003*

Curso J820
Produtividade e Qualidade em Java: Ferramentas e Metodologias

Revisão 1.1

(helder@acm.org)

argonavis.com.br