Sistemas de Controle de Versão

Controle de versão

- **Controle de versão**
  - *"conjunto de práticas cujo objetivo principal é manter controle sobre as modificações efetuadas em um determinado (conjunto) arquivo."*
  - Utilizado no contexto de uma **Gerência de Versões**
  - Aplicado aos arquivos de um **Projeto**.

Sistemas de Controle de Versão (SCV)

- Ferramenta de apoio ao Gerenciamento de Versões
  - De documentos, códigos fontes, etc.

- Permitem
  - Automatizar o processo de controle de versão.
  - Gerenciar o histórico, modificações e diferentes versões;
  - Que várias pessoas trabalhem simultaneamente em um mesmo documento;
  - Controlar as versões através de tags;
  - Visualizar diferenças entre as versões;
  - *merging* entre versões conflituosas;

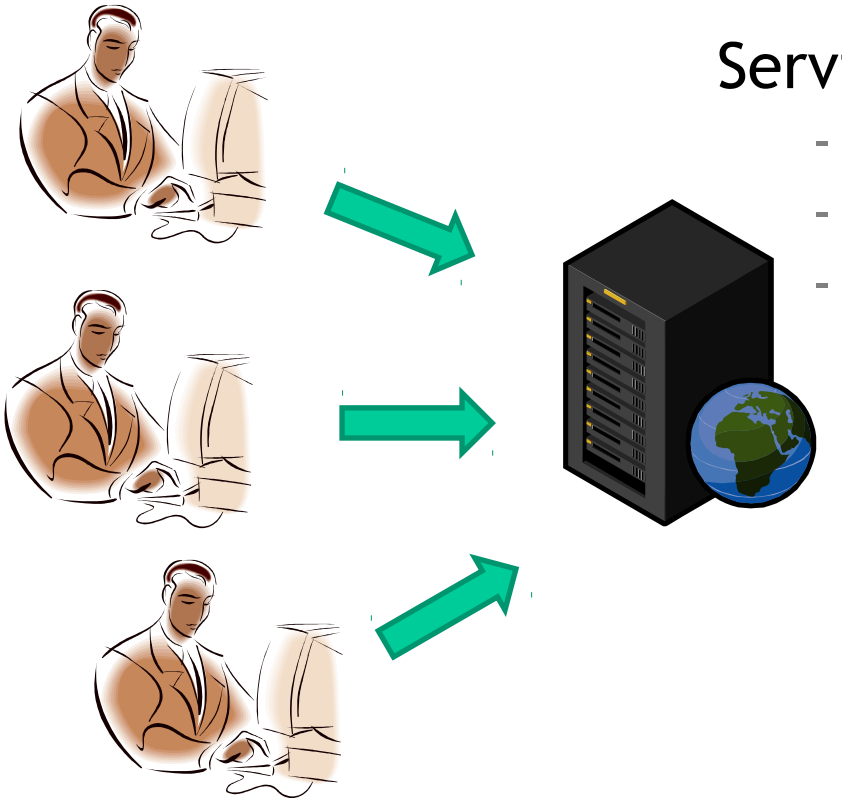
O que não um SCV é?

- Mecanismo para backup
- Ferramenta para a construção de builds
- Substituto para comunicação entre desenvolvedores
  - Conflitos não são resolvidos automaticamente
- Ferramenta de solicitação de mudanças
  - Não controla requisições de mudança

SCVs não são a solução para todos os problemas!

SCVs são utilizados dentro de um contexto maior, e possuem um papel específico.
Veremos outras ferramentas, com outros papeis, ainda neste curso!

Arquitetura Cliente/Servidor



Servidor mantêm

- **Repositório centralizado**
- arquivos a serem controlados,
- histórico de suas mudanças, log, etc.

*- A maioria dos SCVs armazena apenas as diferenças (**deltas**) entre versões sucessivas*
- As somas das diferenças a produzem versão mais nova

Clientes mantêm

- **Área de trabalho** (Workspace)
- cópia dos arquivos do repositório

Principais SCV OpenSource

- Concurrent Versions System (CVS)
  - *Referência para todas as outras ferramentas subseqüentes.*
  - **Porém...**
    - tem perdido espaço para outras ferramentas
- Subversion (SVN): *Substituto melhorado do CVS.*
    - Versionamento de diretórios;
    - Commits atômicos;
    - Uso eficiente de rede.
- Git: SCV **distribuído**
    - Muito rápido
    - Repositórios nunca ficam inconsistentes
    - Sem a necessidade de servidores

Tendências

- **Grande procura pelo SVN e pelo Git**
- CVS está caindo em desuso
- Ferramentas comerciais
  - ClearCase, Perforce, Bitkeeper

continuam a ser adotadas apenas em empresas de Grande Porte.

String de busca: CVS version, SVN version, Git version

Comparação entre alguns SCVs

	CVS	SVN	ClearCase	Git
Licença	Open Source	Open Source	Comercial	
Formato Repositório	Arquivos RCS [1]	BD relacional		BD de objetos
Atomic Commit	Não	Sim	Sim	Sim
Copiar e Renomear Arquivos e Diretórios	Não	Sim	Sim	Sim
Merge Tracking	Não	Sim	Sim	Sim
Tags	Sim	Sim[2]		Sim
Conjunto de Comandos	Simples	Excelente	Excelente	Excelente
Deployment	Bom	Bom	Fraco [3]	Bom
Velocidade	Médio[4]	Muito Bom	Média [5]	Excelente
Portabilidade	Bom	Excelente	Médio	Bom

[1] Arquivos RCS podem ser alterados manualmente quando corrompidos, porém não suportam transações.
[2] Suportado através de copias.
[3] O ClearCase tem uma instalação difícil..
[4] Para suportar segurança, o CVS precisa ser tunelado dentro de outros protocolos.
[5] Servidor e clientes precisam estar na mesma rede para se obter uma performace aceitável.
*Principais Fontes: Wikipedia Comparison e Better SCM Comparison
Comparação detalhada: http://better-scm.shlomifish.org/comparison/comparison.html

Fluxo de Trabalho com SVC

Modelos de “Versionamento”

Lock-Modify-Unlock

- Falsa noção de segurança.
  - Mais problemas do que parece.
- Você só consegue alterar um arquivo se conseguir destravá-lo.
  - Desenvolvedores esquecem arquivos travados freqüentemente!
- Dificulta uso off-line.

Modelos de “Versionamento”

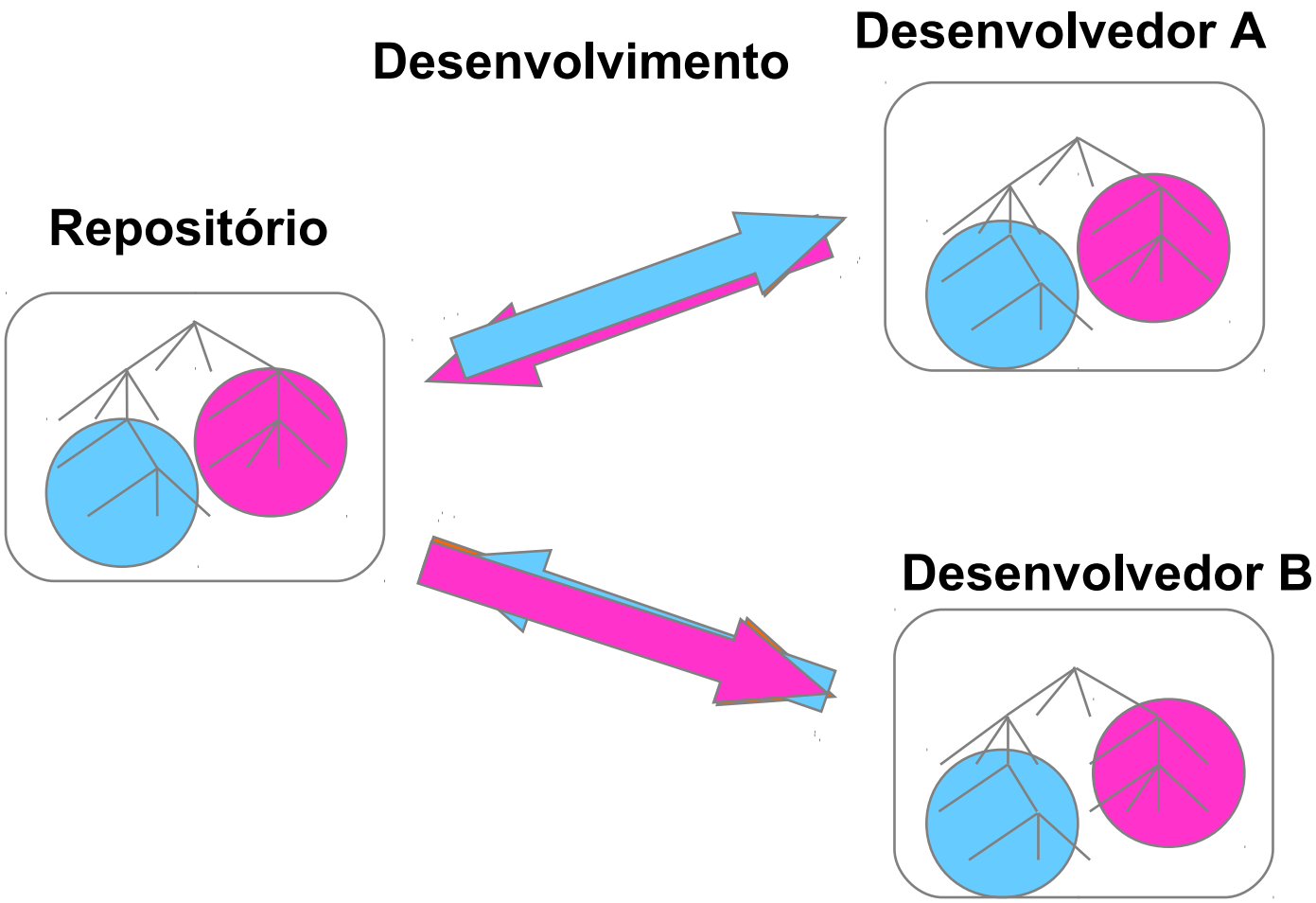
Lock-Modify-Unlock

- Falsa noção de segurança.
  - Mais problemas do que parece.
- Você só consegue alterar um arquivo se conseguir destravá-lo.
  - Desenvolvedores esquecem arquivos travados freqüentemente!
- Dificulta uso off-line.

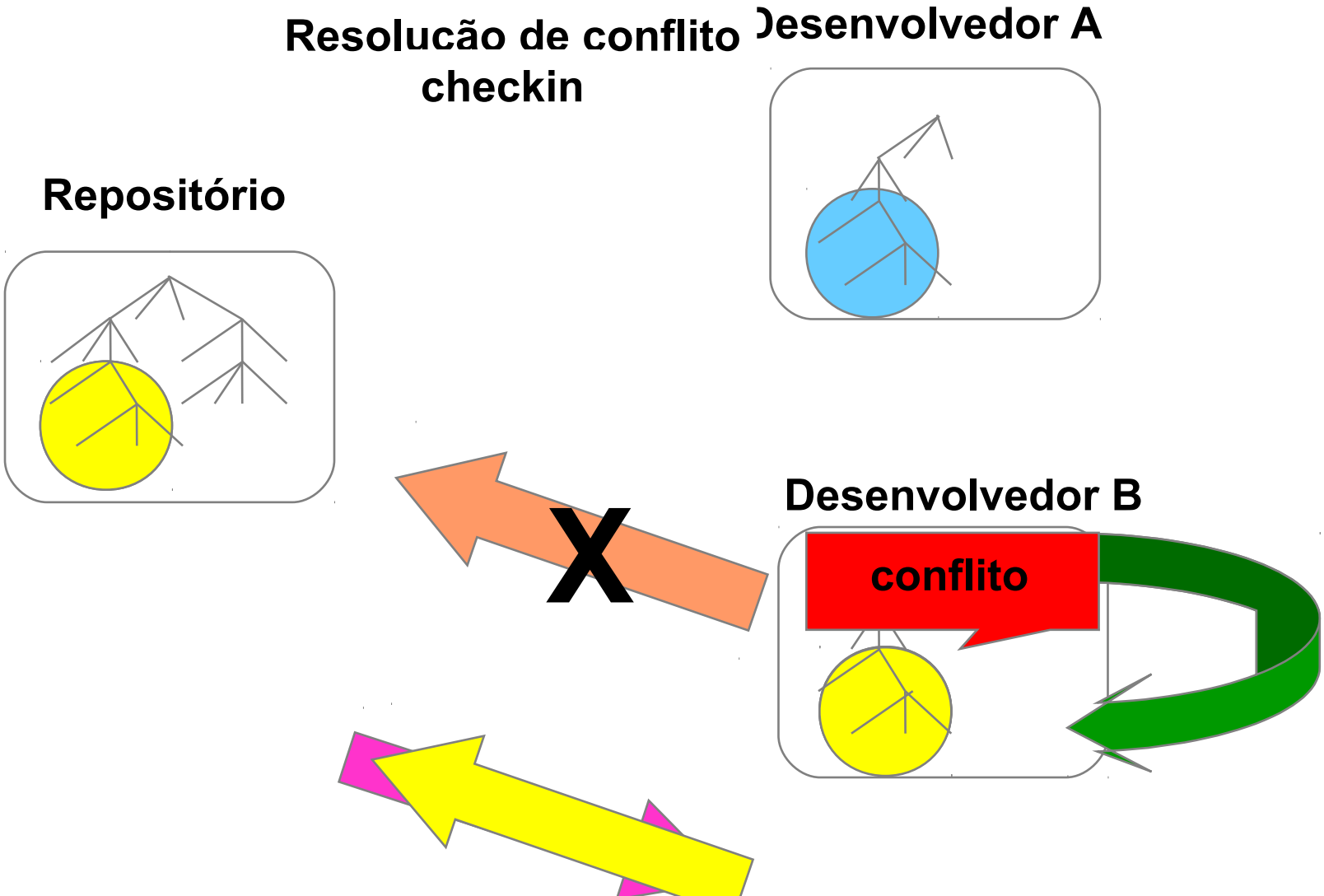
Copy-Modify-Merge

- Método recomendado
- Mais simples e prático.
  - menos problemático do que você imagina
- Desenvolvedores podem trabalhar
  - simultaneamente no mesmo arquivo.
  - Livremente em seu workspace
- Facilita o uso off-line.

Desenvolvimento ideal



Desenvolvimento real



Fluxo de Trabalho

- Comece sem nada.
  - Selecione um projeto em uma codeline (branch ou raiz) e selecione **Checkout;**
    - ou **update** com a versão mais atual do repositório se o projeto já existe localmente.
- Faça as mudanças.
  - Trabalhe localmente com o projeto, salvando as mudanças apenas na sua máquina;
- Sincronize, quando você estiver pronto
  - Update;
  - Examine as mudanças, faça as alterações necessárias;
  - Rode os Testes;
  - Commit;

Integração Continua

Imaturidade no Desenvolvimento

Projetos custam demais, demoram demais, falham demais...

- Desenvolvimento de software continua imaturo.
  - Ele atribui mais responsabilidades ao indivíduo
    - para que ele tenha maior liberdade para exercer sua criatividade.

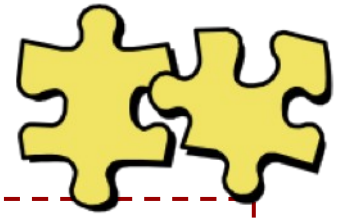
Porem …

Os desafios atuais exigem que o desenvolvimento seja balanceado

- com uma **abordagem mais automatizada** e estruturada
- que otimize a produção e suporte a melhoria continua do produto e do processo.

Atividades de integração

- A integração de partes de um sistema
  - normalmente é apenas perto de sua implantação
    - Problemas são detectados tardiamente.
  - em geral, é feita em freqüência inversamente proporcional à complexidade do sistema
    - Dificuldade em rastrear as causas dos problemas
  - É uma tarefa trabalhosa e sujeita a erros
    - Quanto maior o sistema, mais difícil



*"**Integração** é uma das tarefas de desenvolvimento que **pode se beneficiar**, e muito, de um **processo automatizado e estruturado**."*

Integração Contínua

- Geração freqüente (pelo menos diária) de *builds* do sistema
  - partes do sistema integradas constantemente
  - problemas de integração encontrados logo que introduzidos, na maioria dos casos
- Considerada uma das "melhores práticas" no desenvolvimento de *software*

Há alguns problemas, porém...

Builds

construção complexa

Os Problemas na Geração de *Builds*

- Fazer *builds* do sistema **manualmente**
  - pode ser muito demorado;
  - é difícil de manter;
  - não provê reuso entre equipes
- Gerências de builds
  - builds não são disponíveis para toda a equipe.
  - problemas em builds anteriores são ignorados;
- Dependências dos builds
  - Pode ser difícil saber quais as versões "corretas" dos arquivos utilizados no build;
  - Os pedaços do sistema podem estar em diversos locais diferentes;
  - Alguns arquivos podem ser esquecidos

As Soluções

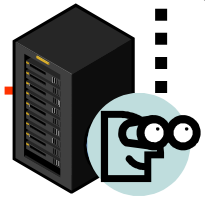
Sistema de Controle de Versões (CVS, por exemplo)

- Manter todos os arquivos em um repositório central
- Controlar o acesso a esse repositório, de modo a garantir a consistência dos artefatos

- Automatizar o processo de geração de *builds*

Build Script

- *Gerenciar a execução dos builds e distribuir seus resultados.*

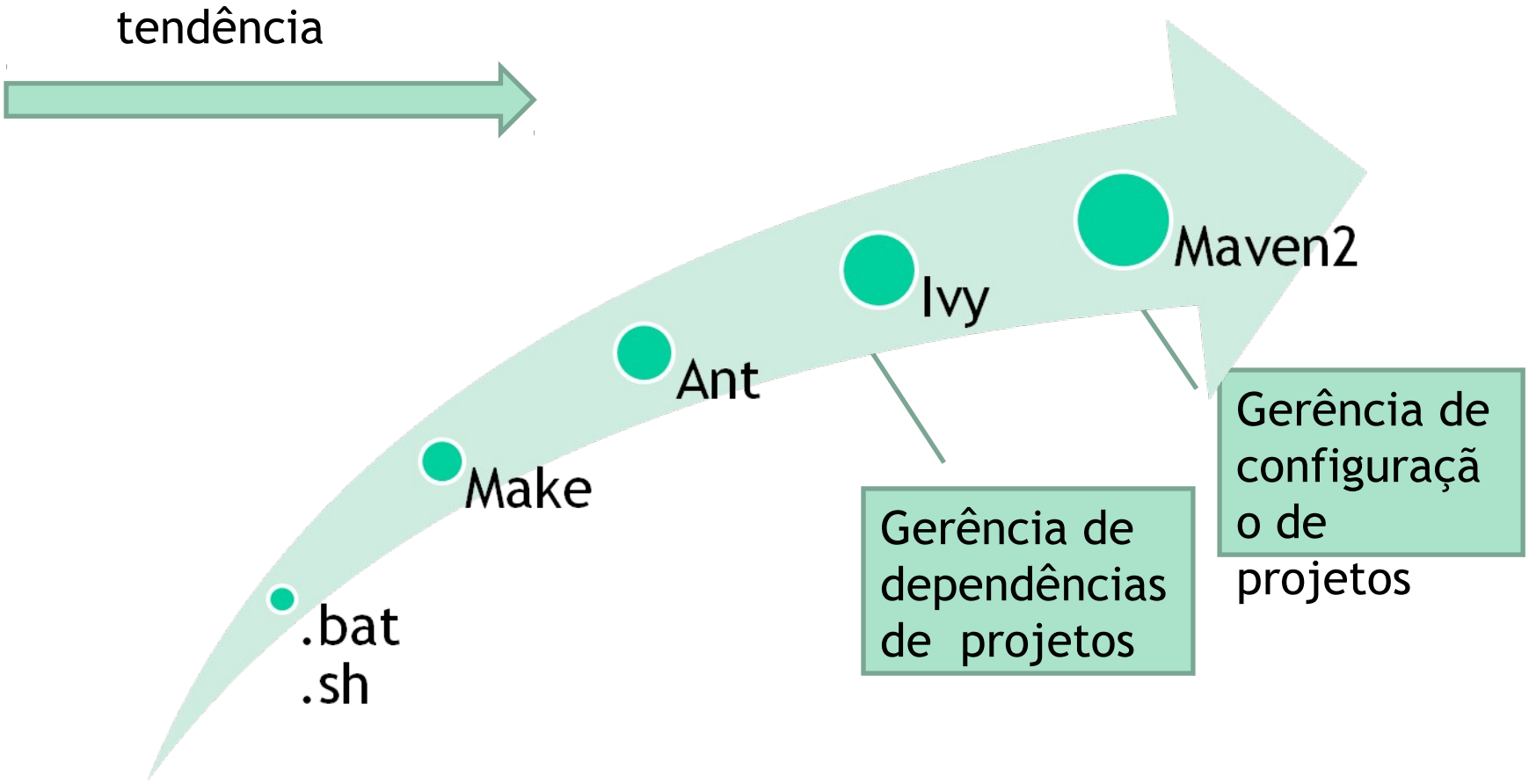


executa

observa

Sistema de Integração Contínua

ferramentas de build [Evolução]



Obs.: Cronologicamente o maven foi criado anteriormente ao Ivy. Porém o primeiro release do Ivy e do maven2 foram lançados na mesma época.

Comparação Maven vs Ant + Ivy

	Critério	Ant + Ivy	Maven
produtividade	**Instalação**	Muito fácil	Muito fácil
	Iniciar um novo projeto	5 min	15 min
	Nova funcionalidade	10 min para cada novo alvo	2 min para cada nova meta
	Aprendizado para um novo desenvolvedor	30 min. Facilidade para se entender e ferramental	2 horas. Pode ser confuso no início.
suporte	**Layout padrão?**	Não (pode ser bom, pois pode alterá-lo como quiser)	Sim (pode ser bom, pois padroniza todos os projetos)
	Suporte a múltiplos projetos?	Sim, mas você tem que criar toda a lógica p/ tratá-los.	Sim, nativo com o Maven Reactor.
	Geração de documentação	Nativamente, não, mas existem plug-ins.	Sim
	Integração com IDEs?	Sim e muito boa	Sim, mas básica
	Gerência de dependências?	Sim, através do plug-in Ivy	Sim, nativo