Gestão de Configuração e Mudanças de Software

Allan Lima

Arquitetura, Design e Implementação de Sistemas para Internet
Pós Graduação
Faculdade 7 de Setembro

arglbr@gmail.com

nov/2015

Integração Contínua de Software

Conceitos

A integração contínua descreve um conjunto de práticas de engenharia de software que acelera a entrega de software, diminuindo o tempo de integração

Conceitos

O que é Integração Contínua?

- Uma prática de desenvolvimento de software onde membros de um time integram seu trabalho frequentemente
- Normalmente cada pessoa integra pelo menos uma vez ao dia, levando a múltiplas integrações por dia
- Cada integração é verificada por uma *build* automatizada (Incluindo testes) para detectar erros de integração o mais rápido o possível
- Essa abordagem leva a uma redução dos problemas de integração e permite que a equipe desenvolva software coeso mais rapidamente

Conceitos

Como tudo começa?

- Quando inicia uma mudança, o desenvolvedor obtém uma cópia da base de código atual em que irá trabalhar
- A medida que o código é alterado por outros desenvolvedores, esta cópia gradualmente deixa de refletir o código do repositório
- Quando o desenvolvedor deseja submeter seu código para o repositório, este deve primeiro atualizar sua cópia para refletir as alterações no repositório
- ...

Conceitos

Como tudo começa?

- Quanto mais mudanças houver no repositório, mais trabalho o desenvolvedor terá antes de submeter suas próprias alterações
- Eventualmente, o repositório pode tornar-se tão diferente da cópia do desenvolvedor que este entra no que se chamar de **inferno de integração**
- Onde o tempo que leva para a integração é maior do que o tempo que levou para fazer as alterações
- Em um pior caso, as mudanças que o desenvolvedor está fazendo podem ter que ser descartadas e o trabalho refeito

Conceitos

Como evitar o retrabalho?

- Integração Contínua é a prática de integração precoce e, muitas vezes, evita as armadilhas do **inferno de integração**
- O objetivo final é reduzir o retrabalho desnecessário e, assim, reduzir custos e tempo
- Devemos procurar as melhores práticas para conseguir a integração contínua
- A automatização é uma das melhores práticas

Como surgiu

Histórico

- Integração Contínua surgiu com a comunidade de Extreme Programming (XP)
- Seus defensores Martin Fowler e Kent Beck foram os primeiros a escrever sobre a integração contínua perto da virada do milênio
- O artigo de Fowler (e depois um livro) é uma fonte importante de informação sobre o assunto
- O livro Extreme Programming Explained (1999, ISBN 0-201-61641-6) de Beck, que é a referência inicial para o XP, também descreve o termo.

Práticas Recomendadas

- A integração contínua se refere à prática de integração frequente de um código com o código que está para ser entregue
- Geralmente no ramo principal do controle de versão, mas isso não é necessariamente uma regra
- O termo **frequência** está aberto à interpretação, mas muitas vezes é entendido como **muitas vezes a cada dia**

Práticas Recomendadas - Resumo

1. Manter o código em um repositório
2. Automatizar a `build`
3. A `build` deve ser auto-testável
4. Submeter (`commit`) o código todos os dias
5. Todo `commit` no ramo principal gera `build`
6. Mantenha a `build` rápida
7. Teste em um clone do ambiente de produção
8. Torne fácil de obter as últimas versões
9. Todos podem ver o resultado da `build` mais recente
10. Automatize a Implantação do Sistema

Práticas Recomendadas

Manter o código em um repositório

- Esta prática preconiza a utilização de um sistema de controle de versão para o código fonte
- Todos os artefatos necessários para construir o projeto devem ser colocados no repositório
- O sistema deve ser capaz de ser construído (`build`) a partir de uma cópia limpa do controle de versão

Práticas Recomendadas

Manter o código em um repositório

- Martin Fowler defende que, mesmo que **ramificações** (branch/fluxos) sejam suportadas pelas ferramentas, sua utilização deve ser **minimizada**
- É preferível que as mudanças sejam integradas ao invés de criar várias versões do software para serem mantidas simultaneamente
- A linha principal deve ser o lugar para a versão de trabalho do software

Práticas Recomendadas

Automatizar a build

- Automação do `build`/integração, muitas vezes inclui a implantação em um ambiente semelhante ao de produção
- Em muitos casos, o `script` de `build` não só compila os binários, mas também gera documentação, páginas do site, estatísticas e os meios de distribuição.
- Exemplos: Windows MSI, RPM, deb, JAR, WAR, EAR e etc

Práticas Recomendadas

A build deve ser auto-testável

- Uma das melhores práticas, certamente a melhor, é o desenvolvimento orientado à testes
- Esta prática consiste em escrever um teste que demonstra como uma funcionalidade deve atuar, e então escrever o código que faz o teste passar
- Uma vez que o código é construído, todos os testes devem ser executados para confirmar se ele se comporta como o esperado

Práticas Recomendadas

Submeter (`commit`) o código todos os dias

- Submeter as alterações no código regularmente, pode reduzir o número de alterações em conflito
- Efetuar `commit` no código de uma semana de trabalho, aumenta o risco de conflito com outras funções e pode ser muito difícil de resolver
- É mais fácil resolver pequenos conflitos, além de motivar os membros da equipe a se comunicarem para solucionar o problema
- É recomendado submeter todas as mudanças pelo menos uma vez por dia e além disso realizar uma `build` noturna

Práticas Recomendadas

Todo `commit` no ramo principal gera `build`

- Todo `commit` deve ser construído para verificar se ele foi integrado corretamente
- Uma prática comum é usar o Ambiente de Integração Contínua, embora isso possa ser feito manualmente
- A integração manual deve ser feita antes do desenvolvedor submeter o código ao sistema de controle de versão
- Para muitos integração contínua é sinônimo de um servidor ou serviço que monitora o sistema de controle de versão atrás de mudanças, em seguida, executa automaticamente o processo de construção
- Este processo pode ser executado manualmente

Práticas Recomendadas

Mantenha a `build` rápida

- A compilação deve ser rápida, pois se houver um problema com a integração, este será rapidamente identificado
- É ter cuidado com a desempenho dos testes automáticos

Práticas Recomendadas

Teste em um clone do ambiente de produção

- Ter um ambiente de teste, com as mesmas características da produção, pode revelar falhas em sistemas testados antes que sejam implantados no ambiente de produção
- Ter um ambiente de testes muito diferente do de produção pode levar à falhas graves
- É aceitável que o ambiente de teste seja menos robusto, porém devemos ter cuidado com a desempenho do sistema na produção

Práticas Recomendadas

Torne fácil de obter as últimas versões

- Tornar as `builds` prontamente disponíveis para o cliente e a equipe de testes pode reduzir a quantidade de retrabalho se for necessário reconstruir uma funcionalidade que não atende aos requisitos
- Além disso, o teste precoce reduz as chances de defeitos chegarem até a implantação
- Encontrar problemas antes da implantação também reduz a quantidade de trabalho necessário para resolvê-los

Práticas Recomendadas

Todos podem ver o resultado da `build` mais recente

- Deve ser fácil descobrir se a `build` está quebrando e quem fez a alteração
- A intenção não é culpar quem cometeu o erro, mas resolver o problema
- É importante resolver o problema enquanto ele é pequeno
- É interessante usar um `site` para aqueles que não estão próximos fisicamente poderem ter uma ideia do estado do projeto

Práticas Recomendadas

Automatize a Implantação do Sistema

- É importante ter `scripts` que permitam facilmente implantar a aplicação dentro de qualquer ambiente
- Uma consequência natural disto é ter `scripts` que permitam implantar o software dentro do ambiente de produção tão facilmente quanto nos outros ambientes
- O software pode não ser implantado no ambiente de produção todos os dias, mas implantações automáticas ajudam tanto a tornar o processo mais rápido quanto a reduzir erros

Benefícios

- Quando os testes de unidade falham, ou um *bug* é descoberto, os desenvolvedores podem reverter o código para um estado livre de erro, sem desperdiçar o tempo de depuração
- Os problemas de integração são detectados e corrigidos continuamente - sem interrupção de último minuto antes de datas de lançamento
- O alerta precoce de código quebrado ou incompatíveis
- O alerta de alterações conflitantes

Benefícios

- Testes de unidade imediatos de todas as alterações
- Disponibilidade constante de uma `build` para propósitos de teste, homologação ou implantação
- O impacto imediato da verificação do código incompleto ou quebrado, é um incentivo para os desenvolvedores aprenderem a trabalhar de forma incremental com ciclos curtos de *feedback*

Benefícios

- Integração Contínua não nos livra dos *bugs*, mas os tornam mais fáceis de encontrar e remover
- *Bugs* destroem a confiança, bagunçam os cronogramas e a reputação
- Com o trabalho em progresso com problemas, fica difícil ter o resto do software funcionando corretamente
- Os *bugs* também são cumulativos: quanto mais tiver, mais difíceis de remover
- As pessoas têm menos energia para encontrar e eliminar erros se existem muitos:
  Fenômeno conhecido como a síndrome da Janela Quebrada

Benefícios

O maior e mais abrangente benefício da Integração Contínua é a redução de riscos. **Martin Fowler**

Por onde começar?

- Seguir o conjunto de práticas
- Não existe uma receita fixa
- Muita coisa depende da natureza do seu ambiente e do seu time
- Um dos primeiros passos é ter uma `build` automatizada
- Ter tudo o que precisa dentro do controle de versão
- Introduzir testes automatizados na sua `build`

Por onde começar?

- Tentar agilizar o **commit build**
- Integração Contínua com uma `build` de poucas horas é melhor do que nada
- Tentar diminuir para o número mágico de 10 minutos
- Ao começar um projeto novo, comece com a I.C. desde o início
- Acima de tudo tenha alguma ajuda:
  Consultoria
  Mentoria
  Estudo dedicado

Concluindo...

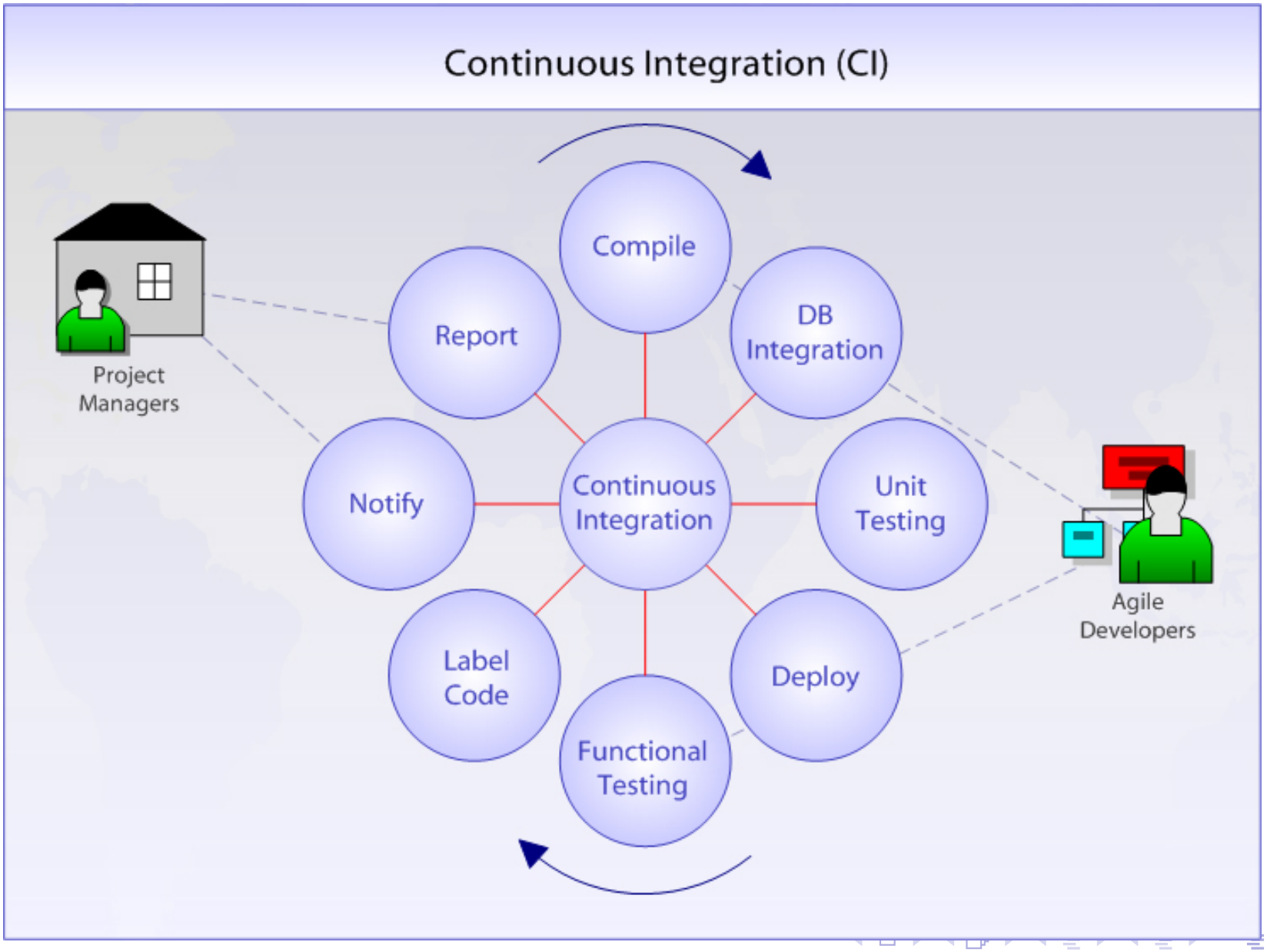
Como vimos...

Build : Consiste da compilação, teste, inspeção e *deployment*, entre outras coisas. É o Software funcionando como uma unidade.

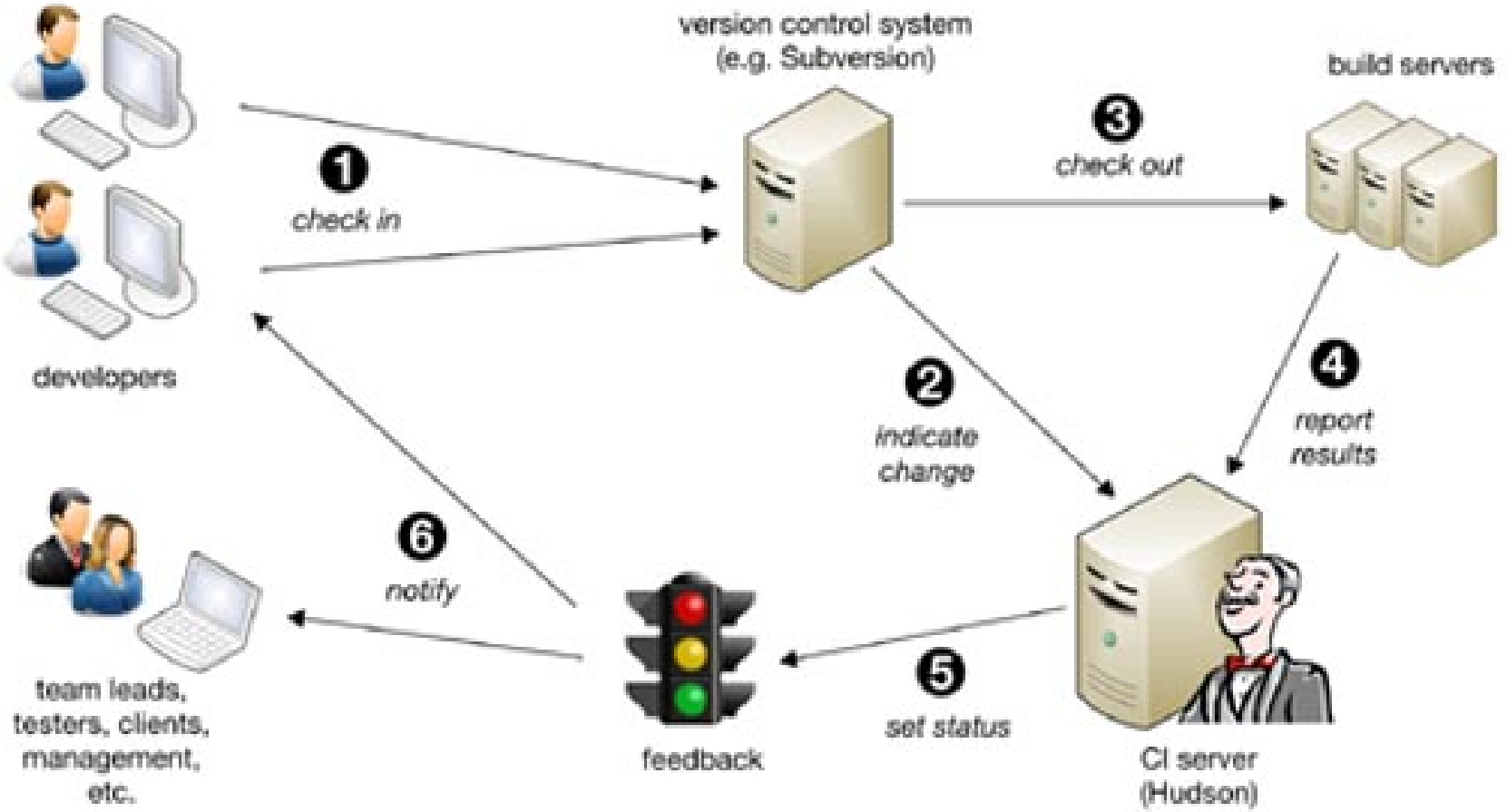
Build Privada : Ao concluir uma tarefa o desenvolvedor deve rodar uma `build` privada (que integra as mudanças do resto do time). A `build` privada é gerada na estação do desenvolvedor.

Servidor de IC : É uma máquina separada que possui a missão de integrar software. A máquina de integração de `build` hospeda o Servidor de I.C.

Concluindo...


Continuous Integration (CI)
Compile
DB Integration
Unit Testing
Deploy
Functional Testing
Label Code
Notify
Report
Continuous Integration
Project Managers
Agile Developers

Concluindo...


version control system
(e.g. Subversion)
build servers
1
check in
3
check out
developers
2
indicate
change
4
report
results
6
notify
team leads,
testers, clients,
management,
etc.
feedback
5
set status
CI server
(Hudson)