# Integração Contínua

Hugo Lopes Tavares

Instituto Federal Fluminense

htavares@iff.edu.br

04 de março de 2010

# *Resumo*

Com a ascensão de metodologias ágeis, entre elas principalmente Extreme Programming, várias práticas de desenvolvimento de software foram amplamente disseminadas e se provaram eficientes. Dentre tais técnicas estão Desenvolvimento Guiado por Testes *(Test-Driven Development)* e Integração Contínua, técnicas estas que se completam. Este trabalho abordará alguns conceitos relacionados a integração contínua e apresentará formas de aplicá-la no dia-a-dia de uma equipe, usando uma abordagem automatizada.

Palavras-chave: Ágil, Integração Contínua, Sistema de Controle de Versão, Test-Driven Development, Refactoring, Extreme Programming

## 0.1 Introdução

Segundo Duvall (2007), o problema de integrar software não é um problema novo, e a medida que a complexidade de um projeto aumenta a necessidade de integrar códigos de diferentes membros de uma equipe cresce. Duvall (2007) diz que um dos pontos de maior risco no ciclo de desenvolvimento de software é a integração de modificações independentes feitas por diferentes membros de uma equipe.

Duvall (2007) salienta que esperar até o fim do projeto pra fazer integração leva a todo tipo de problemas de qualidade, que são custosos e na maioria das vezes trazem atrasos para o projeto. Porém, quando a integração é feita continuamente os riscos são abordados mais rapidamente e em pequenos incrementos antes de ir pra produção. Assim, caso ocorra algum problema, a equipe será alertada rapidamente.

## 0.2 Definição de Integração Contínua

Como o próprio nome já diz, integração contínua significa integrar frequentemente mudanças feitas durante o desenvolvimento de software. Fowler () define integração contínua da seguinte maneira: "uma prática de desenvolvimento de software onde membros de uma equipe integram seu trabalho frequentemente, normalmente cada pessoa integra pelo menos diariamente- levando a múltiplas integrações por dia. Cada integração é verificada por um *build* (incluindo teste) para detecar erros de integração o mais rápido possível."Fowler () também relata que muitas equipes acham que essa abordagem leva a significantes quedas de problemas relacionados a integração e permite que uma equipe desenvolva software coeso mais rapidamente.

## 0.3 Pré-requisitos para Fazer Integração Contínua

Fowler () identifica que integração contínua é simplesmente o processo de integrar código frequentemente - não necessariamente usando um processo automatizado. Porém, como Duvall (2007) discute, os aspectos automatizados de um processo de integração contínua, que segundo o mesmo trazem muito mais benefícios. Porém, uma integração contínua não é possível mesmo quando todo o suporte ferramental existe se a equipe não integrar frequentemente código ao repositório de códigos. Assim, antes de mais nada é necessário que a equipe proponha-se a integrar modificações várias vezes ao dia código que funciona, ou seja, código que satisfaz as necessidades e passa em todos os testes.

### 0.3.1 Repositório de Código

Em qualquer equipe de desenvolvimento é imprescindível o uso de um repositório de códigos e um sistema de controle de versões cuida bem dessa parte - tais como CVS, Subversion, Git, etc - pois além de manter um histórico das alterações do repositório também possibilita baixar versões diferentes dos arquivos.

### 0.3.2 Conjunto Sólido de Testes

Uma peça fundamental dentro do desenvolvimento ágil de software são práticas de técnicas como Test-Driven Development(BECK, 2003) e Behaviour-Driven Development (CHELIMSKY et al., 2010). Usando tais práticas é possível criar um conjunto sólido de conjuntos de testes que tornem possível validar o estado do software frequentemente. É possível identificar problemas de forma automatizada, trazendo benefícios futuros ao projeto, tais como redução no custo da validação do software.

### 0.3.3 Build Automatizado

Um *script de build* automatizado desempenha um papel importantíssimo no ciclo de vida de um projeto, pois com ele é possível que todos consigam instalar o projeto e rodar todos os testes em diferentes máquinas. Usar um script de build significa ter um *script* que cuida de baixar e instalar dependências, compilar códigos-fontes e é responsável por rodar os testes e até mesmo rodar ferramentas de cobertura de teste, notificando caso ocorra erros ou falhas no procedimento.

### 0.3.4 O Ambiente de Produção Deve Ser Clonado

Como descrito por Fowler () os testes devem ser executados numa máquina que possua um ambiente idêntico ao do ambiente de produção. O ponto principal é que usando o mesmo ambiente é possível evitar problemas antes do software ir pra produção. Tanto problemas relacionados a versões de dependências quanto a sistemas operacionais. Essa prática se mostra eficiente pois a equipe não tem surpresas na hora de implantar o sistema no ambiente de produção, pois o ambiente é idêntico ao ambiente em que o software foi desenvolvido e testado.

### 0.3.5 Resumo das Práticas

Usando todas as práticas citadas anteriormente é possível: compartilhar e versionar código entre a equipe, validar o sistema a qualquer momento, instalar e rodar os testes facilmente e evitar surpresas na hora da implantação no ambiente de produção. Vale lembrar que essas práticas são as mais fundamentais quando se fala em integração contínua. Todas devem andar juntas e são um pré-requisito pra qualquer equipe que queira usar integração contínua.

## 0.4 Integração Contínua Síncrona, Assíncrona

Segundo Beck e Andres (2004) há dois modelos de integração contínua: síncrono e assíncrono. No modelo síncrono o *build* é executado na máquina de integração pela equipe assim que um conjunto de alterações está pronto pra ir pro repositório de código. As alterações são baixadas pra máquina, o build é executado e caso seja executado com sucesso, passando em todos os testes e com cobertura de testes satisfatória (caso cobertura seja importante pra equipe), enviado ao repositório de código. O modelo assíncrono é mais comum, segundo Beck e Andres (2004). Nesse modelo os conjuntos de alterações feitas pela equipe são enviados ao repositório de código e uma máquina - chamada máquina de integração - é responsável por executar o *build* sempre que novas alterações forem identificadas no repositório. Esta

mesma máquina é responsável por notificar a equipe se ocorreu algum problema como erros, falhas ou cobertura insatisfatória dos testes - seja por e-mail, mensagem de texto, IRC ou qualquer outro meio de comunicação. Fowler () usa os termos build manual e servidor de integração referindo-se respectivamente a integração síncrona e assíncrona.

Teles () diz que um problema no modelo síncrono é que ele funciona melhor quando a equipe inteira está num mesmo local, pois estabelece-se um acordo sobre a vez de integrar e evita-se que o repositório de código fique com código inconsistente em qualquer momento. O modelo assíncrono, como descrito por Teles (), é recomendado pra projetos onde a equipe está distribuída, ou seja, não estão todos no mesmo local físico. Teles () identifica esta abordagem como ideal para projetos de software livre e open source, projetos estes que na maioria das vezes a equipe está espalhada geograficamente. Comparando os dois modelos de integração contínua Teles () diz usar o modelo síncrono na empresa ImproveIt e diz que o modelo assíncrono é mais arriscado, pois há o risco de em algum determinado momento deixar o repositório de códigos com inconsistências durante algum tempo, pois a equipe nem sempre resolve em tempo hábil o problema, mesmo sendo alertada rapidamente, e pode ser que nesse meio tempo alguém use código inconsistente. Em ambos modelos podem haver uma máquina real ou virtual dedicada a integração das alterações.

No modelo de integração contínua de Duvall (2007) há sempre um servidor de integração contínua responsável por observar as alterações no repositório de códigos, e sempre que acontecer alguma alteração o servidor executará o build e notificará a equipe caso haja inconsistências, da mesma maneira como descrita no processo assíncrono de Beck e Andres (2004). Duvall (2007) deixa claro que o uso de uma máquina dedicada a integração de software é fundamental para um bom processo de integração contínua, não falando hora nenhuma sobre mais de um modelo de integração como os descritos por Beck e Andres (2004) ou Fowler ().

Uma das vantagens de possuir servidores de integração contínua que monitoram o repositório de código é poder ter builds executados em diferentes sistemas operacionais e até mesmo arquiteturas, evitando, assim, o problema do desenvolvedor dizer "mas na minha máquina funciona", como descrito em Kniberg (2007).

## 0.5 Ferramentas

Atualmente há várias ferramentas disponíveis para esta tarefa. Entre as mais famosas estão CruiseControl.rb[1], Hudson[2] e BuildBot[3] - todas ferramentas open-source. Porém, ferramentas mais simples, tais como Make[4], Rake[5], Ant[6] e Maven[7] e Buildout[8] podem ser usadas pra criar builds de forma simples. A diferença das últimas ferramentas em comparação com as primeiras é que essas são mais completas, podendo até mesmo cuidar de envio de e-mails, agendar builds e até comandar diferentes máquinas para executarem builds.

[1] http://cruisecontrolrb.thoughtworks.com/ acesso em 04 de março de 2010
[2] http://hudson-ci.org/ acesso em 04 de março de 2010
[3] http://buildbot.net/trac acesso em 04 de março de 2010
[4] http://www.gnu.org/software/make/ acesso em 04 de março de 2010
[5] http://rake.rubyforge.org/ acesso em 04 de março de 2010
[6] http://ant.apache.org/ acesso em 04 de março de 2010
[7] http://maven.apache.org/ acesso em 04 de março de 2010
[8] http://www.buildout.org acesso em 04 de março de 2010

## 0.6 Conclusão

A prática de usar integração contínua se mostra muito eficaz nas equipes de desenvolvimento e está sendo usada amplamente. Inúmeros projetos open-source usam várias máquinas dedicadas a serem servidores de integração, rodando em muitas vezes sistemas operacionais diferentes ou diferentes versões de softwares, disponibilizando sempre um relatório para a equipe. Em equipes onde há distância geográfica recomenda-se o uso de integração contínua assíncrona e em equipes que trabalham no mesmo espaço físico recomenda-se o uso de integração contínua síncrona. Em ambos casos é importante ter pelo menos uma máquina dedicada a integração que seja um clone do ambiente de produção, pois o quanto mais rápido for o *feedback* em um projeto, melhor.

Em suma, integração contínua resume-se em ter *feedback* rápido e assegura que sempre existirá um software pronto pra ser colocado em produção (obviamente não está com todos os recursos, mas deverá sempre ser usável). Beck e Andres (2004) define *feedback* como um dos valores mais importantes no método *Extreme Programming* (BECK; ANDRES, 2004).

# *Referências Bibliográficas*


BECK, K. *Test-Driven Development by Example*. [S.l.]: Addison-Wesley, 2003.

BECK, K.; ANDRES, C. *Extreme Programming Explained: Embrace Changes*. 2nd. ed. [S.l.]: Addison-Wesley, 2004.

CHELIMSKY, D. et al. [S.l.]: Pragmatic Programmer Bookshelf, 2010.

DUVALL, P. M. *Continuous integration: improving software quality and reducing risk*. [S.l.]: Addison-Wesley, 2007.

FOWLER, M. *Continuous Integration*. Disponível em http://martinfowler.com/articles/continuousIntegration.html, acesso em 03/03/2010.

KNIBERG, H. *Scrum and XP from the Trenches*. [S.l.]: C4Media, 2007. Disponível em http://www.infoq.com/resource/minibooks/scrum-xp-from-the-trenches/en/pdf/ScrumAndXpFromTheTrenchesonline07-31.pdf, acesso em 04/03/2010.

TELES, V. *Integração Contínua*. Disponível em http://improveit.com.br/xp/praticas/integracao, acesso em 25 de fevereiro de 2010.