Integração Contínua

Autor:	Hugo Lopes Tavares
Contato:	htavares@iff.edu.br

Data:	26 de fevereiro de 2010
Palavras-chave:	Ágil, Integração Contínua, Sistema de Controle de Versão, Test-Driven Development, Refactoring, Extreme Programming

Resumo:	Com a ascensão de metodologias ágeis, entre elas principalmente Extreme Programming[1], várias práticas de desenvolvimento de software foram amplamente disseminadas e se provaram eficientes. Dentre tais técnicas estão Desenvolvimento Guiado por Testes *(Test-Driven Development)*[2] e Integração Contínua[3], 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.

Introdução

Segundo Duvall[3], 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. Butcher[4] 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[3] 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.

Definição de Integração Contínua

Como o próprio nome já diz, integração contínua significa integrar frequentemente mudanças feitas durante o desenvolvimento de software. Fowler[5] 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[5] 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.

Pré-requisitos para Fazer Integração Contínua

Fowler[5] identifica que integração contínua é simplesmente o processo de integrar código frequentemente - não necessariamente usando um processo automatizado. Porém, como Duvall[3] 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.

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.

Conjunto Sólido de Testes

Uma peça fundamental dentro do desenvolvimento ágil de software são práticas de técnicas como Test-Driven Development[2] e Behaviour-Driven Development[8]. 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.

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.

O Ambiente de Produção Deve Ser Clonado

Como descrito por Fowler[5] 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.

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.

Integração Contínua Síncrona, Assíncrona

Segundo Beck[1] 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[1]. 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[5] usa os termos build manual e servidor de integração referindo-se respectivamente a integração síncrona e assíncrona.

Teles[6] 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[6], é recomendado pra projetos onde a equipe está distribuída, ou seja, não estão todos no mesmo local físico. Teles[6] 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[6] 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[3] 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[1]. Duvall[3] 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[1] ou Fowler[5].

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 por Kniberg[7].

Ferramentas

Atualmente há várias ferramentas disponíveis para esta tarefa. Entre as mais famosas estão [CruiseControl.rb], [Hudson] e [BuildBot] - todas ferramentas open-source. Porém, ferramentas mais simples, tais como [Make], [Rake], [Ant] e [Maven] e [Buildout] 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.

Conclusões (como será?!)

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[1] define *feedback* como um dos valores mais importantes no método *Extreme Programming*.

Referências

[1]

- Beck, K; Andres, C. Extreme Programming Explained: Embrace Change. 2nd ed. Addison-Wesley (2004)

[2]

- Beck, K. Test-Driven Development by Example. Addison-Wesley (2003)

[3]

- Duvall, Paul M. Continuous integration: improving software quality and reducing risk (2007)

[4]

- Butcher, P. Debug It! (2009)

[5]

- Fowler, M. http://martinfowler.com/articles/continuousIntegration.html (acessado em 25 de fevereiro de 2010)

[6]

- Teles, V. http://improveit.com.br/xp/praticas/integracao (acessado em 25 de fevereiro de 2010)

[7]

- Kniberg, H. XP and Scrum from the Trenches (2007)

[8]

- Chelimsky, D.; Astels, D.; Dennis, Z.; Hellesøy, A.; Helmkamp, B.; North, D. The RSpec Book (2010)

[CruiseControl.rb]

- http://cruisecontrolrb.thoughtworks.com/

[Hudson]

- http://hudson-ci.org/

[BuildBot]

- http://buildbot.net/trac

[Make]

- http://www.gnu.org/software/make/

[Rake]

- http://rake.rubyforge.org/

[Ant]

- http://ant.apache.org/

[Maven]

- http://maven.apache.org/

[Buildout]

- http://www.buildout.org