

ENTREGA CONTÍNUA: UM ESTUDO DE CASO PARA AUTOMATIZAÇÃO DO FLUXO DE IMPLANTAÇÃO DO SISTEMA INTEGRA

Alicenaira Lanes Souza Carneiro de ALMEIDA

Centro de Ensino Superior de Juiz de Fora, Juiz de Fora, MG

Orientador: Evaldo de Oliveira da SILVA

Resumo: Em muitos ambientes de desenvolvimento os processos realizados, entre a solicitação de uma manutenção e a implantação da mesma em produção, são executados manualmente. Esse processo de entrega manual faz com que muitas organizações que produzem *software* tenham um ciclo mais longo para gerar uma nova versão da aplicação. Quanto maior a duração para implantar uma manutenção ou nova funcionalidade em produção, maior será o risco de ocorrerem erros. A Entrega Contínua é uma prática que busca a redução da duração dos ciclos de implantação de sistemas, através da utilização de ferramentas que automatizam o versionamento, a construção e demais processos de desenvolvimento. Este artigo aborda a implantação da Entrega Contínua com o uso de ferramentas *open sources* em um sistema denominado “Novo Integra”, que é desenvolvido e mantido pela equipe de desenvolvimento do Instituto de Ciências Exatas da Universidade Federal de Juiz de Fora. Ao final, uma metodologia de implantação é proposta e os resultados obtidos com essa abordagem são apresentados.

Palavras-chave: Entrega Contínua; Qualidade de *Software*; Automatização de Processos.

1 INTRODUÇÃO

A realização de entregas manuais de *software* pode expor a organização e o usuário final a riscos e gerar um grande desconforto para quem está implantando um sistema em produção. Em muitos ambientes de desenvolvimento os processos realizados, entre a solicitação de uma manutenção e a implantação da mesma em produção, faz com que a duração de um ciclo atinja semanas, meses ou até mesmo mais de um ano. Quanto maior sua duração, maior será o risco de ocorrerem erros durante a implantação. Esse é o cenário atual do sistema “Novo Integra”, que é desenvolvido e mantido pela equipe de desenvolvimento do Instituto de Ciências Exatas (ICE) da Universidade Federal de Juiz de Fora (UFJF).

Com o objetivo de automatizar os processos da entrega de *software*, foi proposta uma metodologia para a implantação da Entrega Contínua, que de acordo com Humble e Farley (2014), “cria um processo de entrega confiável, previsível e passível de repetição, que, por sua vez, gera grandes reduções no tempo de ciclo e entrega novas funcionalidades e correções aos usuários rapidamente”. O estudo de

caso realizado aponta quais foram as ferramentas *open sources* utilizadas e os resultados obtidos com a implantação.

A automatização dos processos diminui o tempo entre um código pronto e seu uso pelos usuários finais. Assim, o risco associado à entrega tem uma redução significativa, permitindo a reversão de mudanças com facilidade e rápida obtenção de *feedback* (SETE, 2013).

Essa automatização pode ser obtida através da integração de todas as atividades da entrega de *software*. Entre elas está o Controle de Modificações, o Controle de Versões, o Controle de Gerenciamento de Construção, os Testes Automatizados, a Análise Estática do Código, a Gerência de Dependências, a Gerência de Artefatos e a Gerência de Infraestrutura. Para que essas atividades sejam transformadas em uma etapa normal e contínua é fundamental que todos os membros da equipe trabalhem em conjunto, isso inclui desde desenvolvedores e testadores, até times de implantação e operação (SATO, 2014).

Este artigo está organizado da seguinte forma: a seção 2 apresenta a fundamentação teórica com as principais técnicas utilizadas na Entrega Contínua; a seção 3 apresenta o cenário inicial do estudo de caso, sugere uma metodologia para a implantação da Entrega Contínua, indica quais ferramentas *open sources* podem ser utilizadas e apresenta os resultados obtidos com essa abordagem; finalmente, a seção 4 apresenta as considerações finais e trabalhos futuros.

2 REFERENCIAL TEÓRICO

A Entrega Contínua (*Continuous Delivery*) surgiu para reduzir o custo, o tempo e o risco da entrega de *software*. Para isso os processos devem ser amparados por ferramentas e *scripts* de automatização que garantam que o *software* estará pronto para ser implantado em produção a qualquer momento (HUMBLE; FARLEY, 2014).

Para apoiar o processo de Entrega Contínua, a Gerência de Configuração de *Software* (GCS) fornece técnicas, processos e ferramentas para automatizar e implantar *software* em ciclos curtos. De acordo com Dantas (2009), os principais sistemas da GCS são o Controle de Modificações, o Controle de Versões e o Controle de Gerenciamento de Construção. A seguir serão detalhadas as principais atividades da GCS e sua importância no processo de entrega de *software*.

2.1 GERÊNCIA DE CONFIGURAÇÃO DE SOFTWARE

Segundo Estublier (2000) a GCS é definida como uma área da Engenharia de *Software* (ES) responsável pelo controle da evolução de sistemas complexos permitindo contribuir com a ES no desenvolvimento e manutenção das aplicações, com a finalidade de possibilitar a rastreabilidade das modificações realizadas.

A configuração de *software* é composta por um conjunto de itens de configuração. Entre eles estão documentos, programas, dados e ambiente de desenvolvimento. Todos esses itens devem ser controlados pela GCS, que aplicada ao longo de todo o ciclo de vida da aplicação. Seu objetivo é identificar, controlar, auditar e relatar modificações que ocorrem durante o desenvolvimento e manutenção do *software*, evitando assim a perda de controle do projeto (PRESSMAN, 2002).

De acordo com Dantas (2009), “sob a perspectiva de desenvolvimento, a GCS abrange três sistemas principais: Controle de Modificações, Controle de Versões e Controle de Gerenciamento de Construção”.

O objetivo do Controle de Modificações é garantir a qualidade e a consistência dos itens de configuração de *software*, à medida que as modificações são feitas. Sua utilização permite acompanhar e gerenciar todas as alterações realizadas durante o ciclo de vida da aplicação (SALES et al, 2010).

Para controlar as modificações dos itens e obter o histórico de alterações, os mesmos devem ser armazenados em um repositório. O Controle de Versões é composto por procedimentos e ferramentas que permitem a administração desse repositório e o versionamento de *software* (GARCIA, 2013).

A construção de *software*, também conhecida como *build*, corresponde ao processo de compilação, teste e empacotamento do sistema, sua automatização pode ser realizada através do Controle de Gerenciamento de Construção. Além dessas etapas, o processo de construção pode incluir o Gerenciamento de Dependências e a execução de ferramentas de Análise Estática do Código, a importância dessas etapas será esclarecida mais adiante. Após a execução de um *build* obtém-se um ou mais arquivos binários, também conhecidos como artefatos, que estarão prontos para serem implantados em produção (SATO, 2014).

Quando mais de uma pessoa trabalha em um mesmo projeto, frequentemente as mudanças realizadas por um membro impactam os demais, podendo conflitar entre si e gerar erros durante a construção ou execução da aplicação. Dessa forma,

além de realizar o controle das mudanças, de versões e o gerenciamento da construção do software, é muito importante que o trabalho de todos os envolvidos seja integrado para a obtenção de um produto coerente (LOPES, 2011).

A Integração Contínua é uma atividade da GCS que consiste em diminuir o ciclo de integração, tornando-o o mais frequente possível. Além disso, um *feedback* contínuo pode ser obtido através de práticas e ferramentas que auxiliam na verificação e correção de erros de integração assim que eles são introduzidos, quando ainda é barato corrigi-los (LOPES, 2011).

O *pipeline* de implantação, que será apresentado mais a frente, é utilizado para levar o princípio de IC à sua conclusão lógica. Ao combiná-lo com o uso correto da GCS e um alto grau de testes automatizados é possível realizar entregas em ambiente de teste, desenvolvimento ou produção, a partir de um clique em um botão.

2.2 GARANTIA E CONTROLE DA QUALIDADE DE SOFTWARE

De acordo com Humble e Farley (2014), testar é uma atividade multifuncional que deve ser executada continuamente desde o começo do projeto, por todos os integrantes do time. Para garantir a qualidade, a cada mudança realizada na aplicação, em sua configuração ou no ambiente de *software*, devem ser escritos testes em múltiplos níveis (unitários, de componentes e de aceitação) e incluídos no *pipeline* de implantação para que sejam executados.

A execução sistemática de testes tem o objetivo específico de encontrar e remover o maior número possível de erros, evitando assim que eles sejam encontrados pelo cliente. Os testes não garantem que o código não possui erros, no entanto, são essenciais para verificar se uma mudança em uma parte do *software* não afetou outras que dependem direta ou indiretamente dela (PRESSMAN, 2002).

Para garantir que o software entregue ao cliente possui alta qualidade, além da execução de testes automatizados, é necessário realizar o controle da qualidade de código através da análise estática do mesmo, que é definida por Terra e Bigonha (2008) como “um dos instrumentos conhecidos pela engenharia de software para a mitigação de erros, seja por sua utilização para a verificação de regras de estilos, para a verificação de erros ou ambos”.

A análise estática do código é muito importante para o controle de qualidade, no entanto, muitos projetos podem evoluir para uma massa de código impossível de

ser mantida, caso não sejam divididos em componentes menores quando isso ainda é possível.

2.3 GERÊNCIA DE DEPENDÊNCIAS E DE ARTEFATOS

A divisão de uma aplicação em uma coleção de componentes com baixo acoplamento e com um bom encapsulamento além de representar uma boa arquitetura também permite colaboração mais eficiente e *feedback* mais rápido em sistemas de grande porte (SATO, 2014).

A diferença de um *software* baseado em componentes é o fato de que sua base de código é separada em porções menores, que possuem interações limitadas e bem definidas. A partir do momento que uma aplicação possui componentes significa que depende dos mesmos para funcionar corretamente. Essa relação é conhecida como dependência. Além dos componentes, uma aplicação pode depender de pacotes que são desenvolvidos, disponibilizados e controlados por terceiros, que são conhecidos como bibliotecas (MURTA, 2006).

A Gerência de Dependências é responsável por recuperar dependências com versões específicas de um repositório da Internet ou de um repositório de artefatos da organização. Quando a organização possui um repositório de artefatos próprio, é possível realizar a Gerência de Artefatos para manter, controlar e disponibilizar bibliotecas e componentes. Dessa forma é possível reduzir o volume de downloads de bibliotecas da internet e conseqüentemente o tráfego na rede (SATO, 2014).

O uso da abordagem baseada em componentes, além de incentivar o reuso, contribui para a colaboração entre grandes equipes. Para alcançar entrega eficiente e *feedback* rápido, é fundamental o uso de componentes, de *pipelines* baseados em dependências, de um gerenciamento eficiente de artefatos, além de um ambiente estável que esteja preparado para a implantação.

2.4 GERÊNCIA DE INFRAESTRUTURA

Um ambiente é formado por um conjunto de recursos necessários para que a aplicação funcione de forma correta. Esses recursos são obtidos através da configuração de hardware dos servidores, da infraestrutura de rede que os conecta e da configuração do sistema operacional e *middleware*. A infraestrutura representa todos os ambientes em conjunto com todos os serviços que os apoiam, isso inclui

servidores de DNS, os firewalls, os roteadores, os repositórios de controle de versão, o armazenamento, as aplicações de monitoramento, os servidores de e-mail, entre outros (HAMBLE; FARLEY, 2014).

Os princípios da Gerência de Infraestrutura incluem a especificação do estado desejado da infraestrutura por meio de configuração mantida no controle de versão; correção automática da infraestrutura até chegar ao estado desejado; e uso de instrumentação e monitoramento para saber o estado atual da infraestrutura. Quando não é realizada de forma correta, resulta em atrasos, perda de eficiência no desenvolvimento e aumento de custo continuado. Sua responsabilidade é preparar e gerenciar ambientes de homologação e produção presentes no Pipeline de Implantação (SATO, 2014).

2.5 O PIPELINE DE IMPLANTAÇÃO

O *pipeline* de implantação é uma implementação de ponta a ponta da automatização do processo de compilação, testes e implantação, que permite às equipes de teste e operação implantar a aplicação em ambientes de teste, de homologação e de produção com o apertar de um botão. Aos desenvolvedores, é permitida a visualização dos estágios do processo de entrega alcançados por cada versão do *software* e dos problemas encontrados em cada uma delas. Além disso, proporciona ao gerente a verificação e monitoração de métricas fundamentais como tempo de ciclo, taxa de transferência e qualidade de acesso, permitindo a visibilidade sobre o processo de entrega que torna possível a identificação, otimização e remoção de gargalos. Dessa forma, obtém-se um processo de Entrega Contínua que além de mais rápido, também é mais seguro (HUMBLE; FARLEY, 2014).

O principal objetivo do *pipeline* de implantação é permitir que todos os envolvidos no processo de entrega tenham visibilidade sobre o progresso dos processos, dessa forma, torna-se possível identificar quais mudanças quebraram a aplicação e quais se tornaram versões candidatas apropriadas para a execução de testes manuais e entrega. Quando uma versão passa por todo o *pipeline* com sucesso, significa que ela foi submetida a testes automatizados e manuais em ambiente similares aos de produção. Essa é a garantia de que está pronta para ser colocada em produção com um simples apertar de um botão (HUMBLE; FARLEY, 2014).

3 METODOLOGIA PARA IMPLANTAÇÃO DA ENTREGA CONTÍNUA

O “Integra” que é desenvolvido e mantido pela equipe de desenvolvimento de *software* do Instituto de Ciências Exatas (ICE) da Universidade Federal de Juiz de Fora (UFJF), pode ser acessado através do endereço <http://integra.ice.ufjf.br/>. De acordo com o próprio site, “o projeto consiste na criação de uma plataforma, que pode ser vista como uma forma de se abstrair a complexidade da união de dados providos de sistemas diferentes através de interfaces de gerenciamento de recursos”.

O sistema interno do ICE foi criado em 1999 e com o passar dos anos notou-se que o mesmo havia se tornado um sistema legado. Devido ao alto acoplamento, foi decidido que um novo sistema deveria ser criado para substituir o antigo. Em 2013 o “Novo Integra” começou a ser desenvolvido. O lançamento da versão inicial está previsto para junho de 2014.

Entre as suas principais funcionalidades está o controle de salas, marcação de aulas, grupos de discussão, eleições, e-mail institucional, matriz curricular, eventos, grade de horário, cadastro de computadores à rede sem fio e o Plano de Expansão e Reestruturação (REUNI).

O “Novo Integra” conta com as tecnologias Java, ZK Framework, Maven, Hibernate e Spring e com as ferramentas NetBeans, Mysql Workbank e Tortoise SVN. Além disso, faz integração com o Sistema Integrado de Gestão Acadêmica da UFJF (UFJF/SIGA) e com o Google.

Atualmente a equipe é formada por três funcionários efetivos (estando um deles afastado para mestrado) e mais dois estagiários. O método de desenvolvimento ágil Kanban é utilizado para apoiar o desenvolvimento e a gerência de modificações.

Devido ao tamanho da equipe, existe uma grande dificuldade para conciliar o desenvolvimento do novo sistema e as manutenções do sistema antigo. A busca pela automatização do processo de entrega do “Novo Integra” começou com o objetivo de reduzir o tempo gasto com tarefas manuais, garantir a qualidade da entrega para evitar o retrabalho e dessa forma aumentar a produtividade da equipe.

O estudo de caso, previamente autorizado pelo Diretor responsável pelo ICE, apresenta os resultados obtidos com o uso da metodologia sugerida no artigo para a automatização do fluxo de implantação do “Novo Integra”. Após a realização da pesquisa sobre Entrega Contínua, foi proposto um modelo que pode ser aplicado em

outros projetos. Através de sua utilização, foram escolhidas as ferramentas que são responsáveis pela automatização de cada um dos processos do estudo de caso. A metodologia foi dividida nos seguintes procedimentos:

- Definição de qual ferramenta será utilizada para acompanhar a evolução do desenvolvimento de *software*;
- Determinação do Sistema de Controle de Versão a ser utilizado;
- Identificação das dependências do projeto e definição de como serão gerenciadas;
- Verificação de qual será o servidor utilizado para a integração entre os módulos dos projetos;
- Definição dos mecanismos que serão utilizados para analisar a qualidade do código;
- Determinação dos tipos de testes necessários para verificar a execução das funcionalidades do projeto;
- Levantamento das configurações dos ambientes para definição de um gerenciador de infraestrutura;
- Estabelecimento de uma solução para a implantação da aplicação em ambiente de produção.

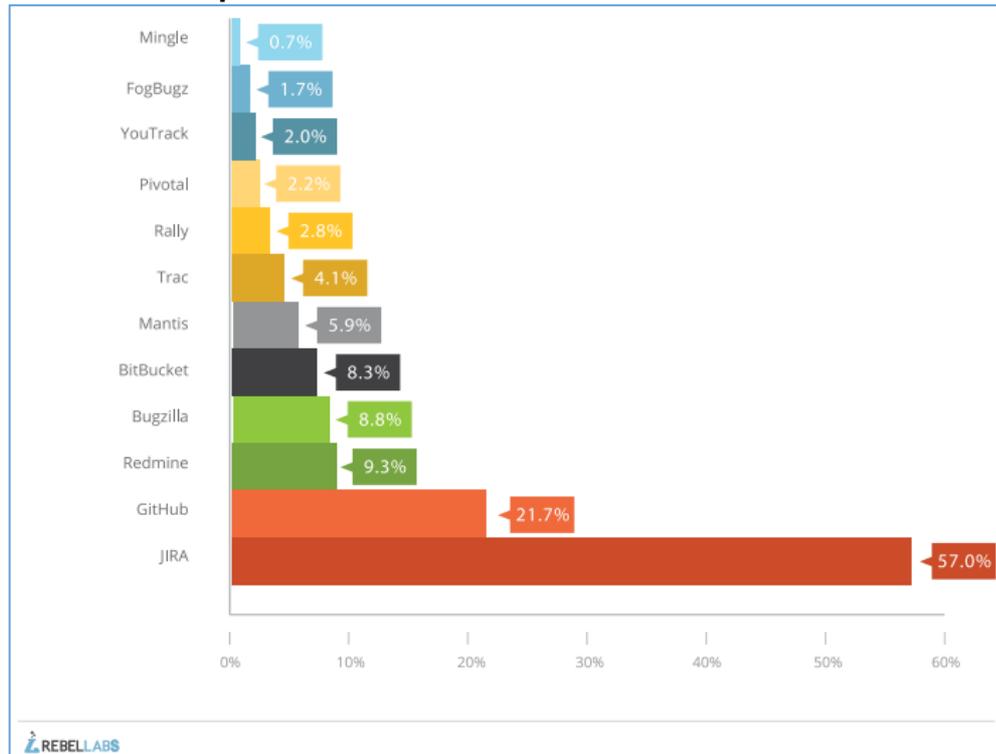
Para fortalecer o modelo sugerido para a implantação da Entrega Contínua, no detalhamento de cada procedimento serão apresentados gráficos que indicam a popularidade de cada ferramenta que pode ser utilizada. Além disso, serão apontadas quais são as opções *open sources* e dentre elas quais foram utilizadas na automatização do fluxo de implantação do “Novo Integra”.

3.1 DEFINIÇÃO DE QUAL FERRAMENTA SERÁ UTILIZADA PARA ACOMPANHAR A EVOLUÇÃO DO DESENVOLVIMENTO DE SOFTWARE

Acompanhar a evolução do desenvolvimento de *software* significa organizar os requisitos em entregas, designar um desenvolvedor para o cumprimento da tarefa e manter um histórico com a descrição do que foi realizado durante o desenvolvimento.

Ferramentas de Gerência de *Issues* (ou requisições de mudanças) atendem aos requisitos citados acima e incluem a gestão de novas solicitações, de mudanças e de defeitos. A Figura 1 aponta quais são os Gerenciadores de *Issues* mais utilizados.

FIGURA 1
Popularidade dos Gerenciadores de Issues



Fonte: White e outros (2013)

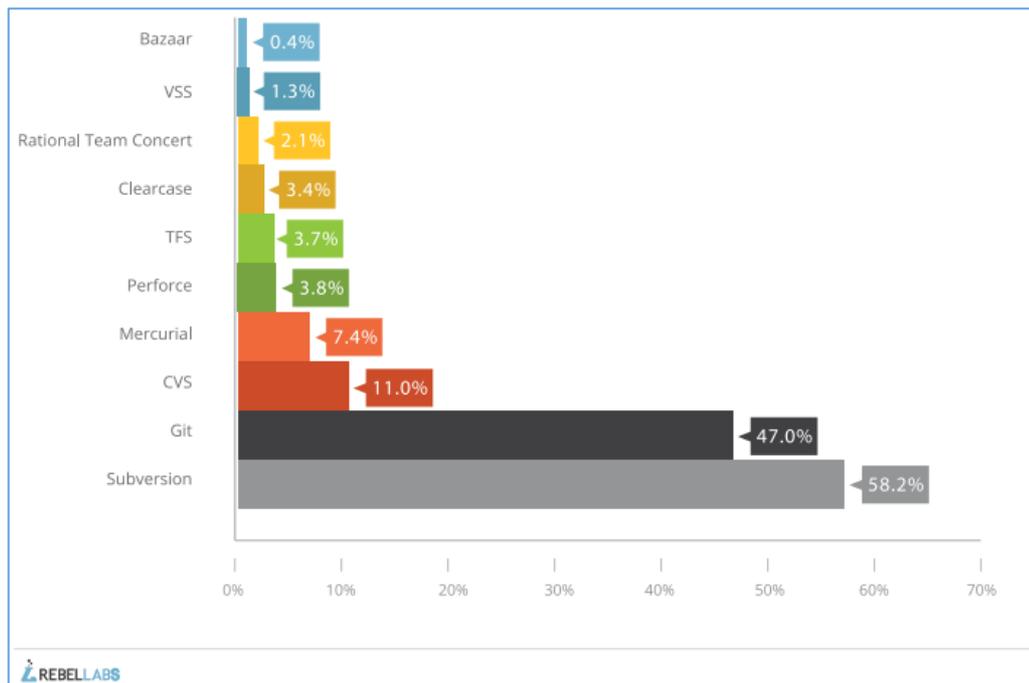
São ferramentas *open sources* o Trac, o Mantis, o Bugzilla e o Redmine. Entre esses, o Redmine (www.redmine.org) é o mais utilizado pelos entrevistados. A equipe de desenvolvimento já utilizava o mesmo, a mudança realizada na implantação da Entrega Contínua foi a integração com o SVN e o Jenkins. Além disso, a instalação do *plugin* ekanban que permite trabalhar com o método de desenvolvimento ágil Kanban, que é praticado pela equipe.

3.2 DETERMINAÇÃO DO SISTEMA DE CONTROLE DE VERSÃO A SER UTILIZADO

Um sistema de controle de versão deve armazenar o código fonte e todos os itens de configuração de *software*. Além disso, deve permitir o acesso a versões anteriores, o compartilhamento entre equipes e a recuperação de informações sobre as alterações através de históricos.

A Figura 2 mostra a popularidade dos Sistemas de Controle de Versão. Entre eles o Mercurial, o CVS, o Git e o Subversion são *open sources*.

FIGURA 2
Popularidade dos Sistemas de Controle de Versão



Fonte: White e outros (2013)

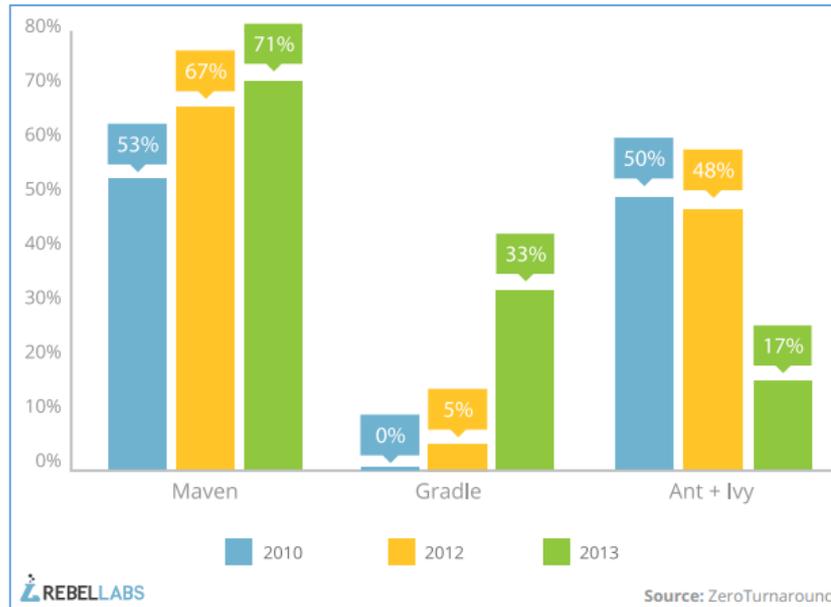
O Subversion (<http://subversion.apache.org/>), também conhecido como SVN é utilizado tradicionalmente em projetos Java e, de acordo com o gráfico, continua sendo o mais utilizado. Apesar da existência de sistemas mais novos, como o Git (<http://git-scm.com/>), o Subversion foi mantido, pois atende a todas as necessidades do projeto e a equipe já domina o seu uso.

3.3 IDENTIFICAÇÃO DAS DEPENDÊNCIAS DO PROJETO E DEFINIÇÃO DE COMO SERÃO GERENCIADAS

A identificação das dependências é necessária para levantar quais modificações serão realizadas durante a implantação das ferramentas responsáveis pelo seu gerenciamento. O uso de uma ferramenta de *build* com suporte a gerenciamento de dependências, em conjunto com um servidor de gerenciamento de artefatos permite a padronização de bibliotecas e componentes do projeto. Além disso, torna possível a execução do *build* a partir de um único comando.

Entre os sistemas *open sources* que possibilitam o *build* automatizado e a gerência de dependências estão o Maven, o Gradle e o Ant (em conjunto com o Ivy). A Figura 3 apresenta a comparação da popularidade dessas ferramentas, considerando o período que vai do final de 2010 até meados de 2013.

FIGURA 3
Popularidade das ferramentas de *build* – Final de 2010 a meados de 2013



Fonte: White e outros (2014)

No início do estudo de caso o Maven (<http://maven.apache.org/>) já estava em uso e foi mantido, pois além de permitir a gestão de dependências e a criação de componentes, também possibilita a automatização do *build* de forma independente, isto é, sem precisar interagir com outras ferramentas.

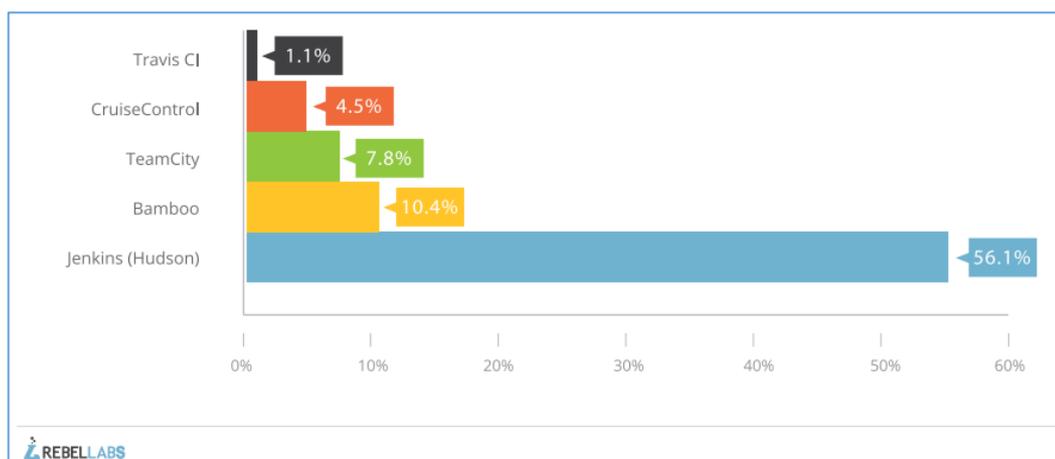
Entre os servidores *open sources* de gerenciamento de artefatos estão o Nexus e o Artifactory. O Nexus está sendo estudado para implantação futura.

3.4 VERIFICAÇÃO DE QUAL SERÁ O SERVIDOR UTILIZADO PARA A INTEGRAÇÃO ENTRE OS MÓDULOS DOS PROJETOS

A automatização de processos por si só não é suficiente. É fundamental possuir um *feedback* que indique se as alterações realizadas em qualquer um dos componentes, módulos ou serviços não quebraram o funcionamento de outro ponto da aplicação.

Servidores de IC possuem a função de integrar todas as alterações realizadas no repositório e podem enviar notificações para os responsáveis em caso de erros. A Figura 4 mostra a popularidade dos servidores de IC. Entre eles o Jenkins (Hudson) é a única opção *open source*.

FIGURA 4
Popularidade dos Servidores de Integração Contínua



Fonte: White e outros (2013)

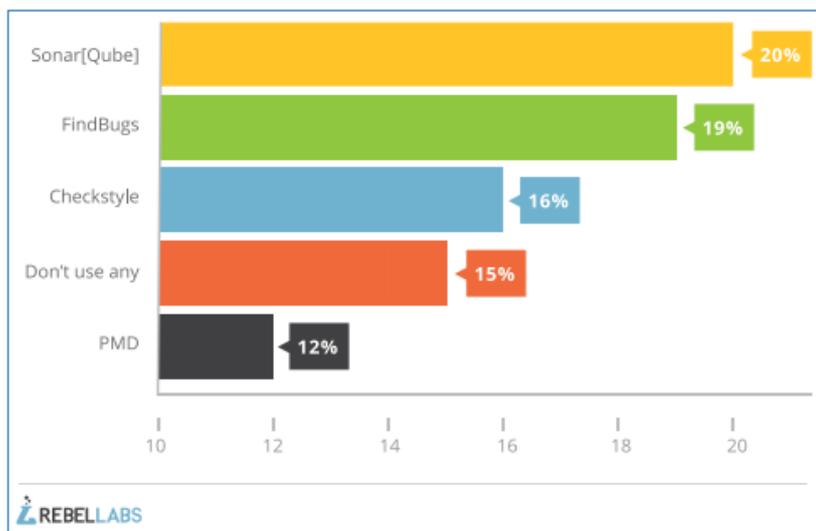
O Jenkins (<http://jenkins-ci.org/>) possui interface web com mais de 600 *plugins* para customização e possibilita a integração com tecnologias que permitem a compilação, análise, testes e implantação das aplicações. É responsável por interligar as ferramentas utilizadas para a entrega de *software*.

No início do estudo de caso o Jenkins já estava em uso, no entanto, era utilizado apenas para integração e verificação da qualidade através de *plugins*. Durante a implantação foram criados e configurados *jobs* específicos para a compilação, análise estática de código, testes e implantação dos componentes, serviços e módulos que são utilizados no *pipeline* de implantação.

3.5 DEFINIÇÃO DOS MECANISMOS QUE SERÃO UTILIZADOS PARA ANALISAR A QUALIDADE DO CÓDIGO

As ferramentas para a Análise Estática de Código utilizam métricas para verificar violações e a evolução da qualidade estrutural do código. A Figura 5 mostra a popularidade das ferramentas utilizadas para esse fim, todas as ferramentas apresentadas são *open sources*.

FIGURA 5
Popularidade das ferramentas de análise estática de código



Fonte: Shelajev e outros (2014)

O Jenkins possibilita a análise da qualidade de código através da instalação de vários *plugins* que geram relatórios e gráficos durante a compilação da aplicação. Entre eles estão o FindBugs (<https://wiki.jenkins-ci.org/display/JENKINS/FindBugs+Plugin>), o CheckStyle (<https://wiki.jenkins-ci.org/display/JENKINS/Checkstyle+Plugin>) e o PMD (<https://wiki.jenkins-ci.org/display/JENKINS/PMD+Plugin>). Após a realização de vários testes, observou-se que o tempo gasto na compilação com a geração desses relatórios chegou a ser treze vezes maior que o tempo gasto para uma compilação que não envolvia essa análise.

A melhor solução encontrada foi a implantação do Sonar (<http://www.sonarqube.org/>), que é uma ferramenta dedicada ao controle e medição da qualidade do código, que também pode ser customizada com a instalação e configuração de *plugins* e integrada ao Jenkins. Suas métricas podem ser customizadas de acordo com a necessidade de cada projeto e é possível definir um limite mínimo aceitável de incidentes que, se ultrapassado, impedirá que a compilação seja realizada com sucesso.

3.6 DETERMINAÇÃO DOS TIPOS DE TESTES NECESSÁRIOS PARA VERIFICAR A EXECUÇÃO DAS FUNCIONALIDADES DO PROJETO

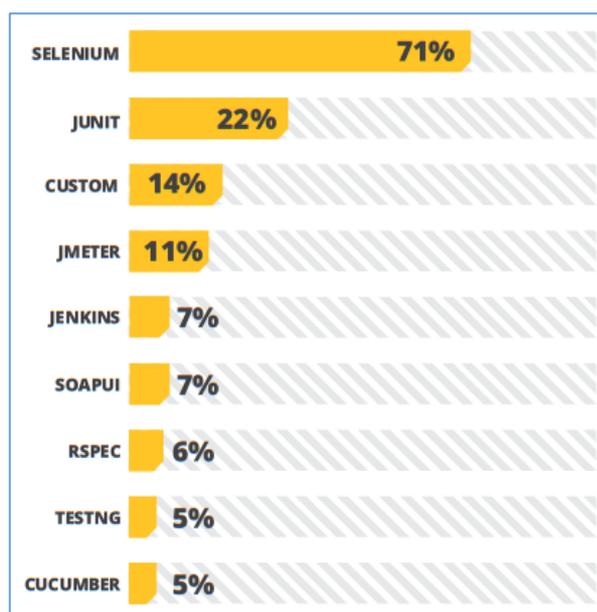
O *build* de uma aplicação garante que não ocorreram erros durante a construção do *software*, no entanto, para entregar valor ao cliente é necessário

atender aos requisitos especificados. Os testes automatizados são os responsáveis por garantir que quaisquer problemas que comprometam o cumprimento dos requisitos sejam identificados enquanto o custo para corrigi-los ainda é baixo, por isso devem ser bem definidos. Dessa forma, obtém-se a garantia de que o *software* está funcionando como deveria, isso significa menos defeitos, menor custo com manutenção e melhor reputação com o cliente.

Ferramentas para diferentes tipos de testes podem ser escolhidas de acordo com o escopo e a necessidade de cada projeto. A Figura 6 exibe a popularidade das ferramentas de automatização de testes. Todas as ferramentas listadas são *open sources*.

As ferramentas escolhidas inicialmente para automatização de testes do “Novo Integra” foram a JUnit (<http://junit.org/>) para a criação de testes unitários e a Selenium (<http://docs.seleniumhq.org/>) para a criação de testes funcionais.

FIGURA 6
Popularidade das ferramentas de automatização de testes



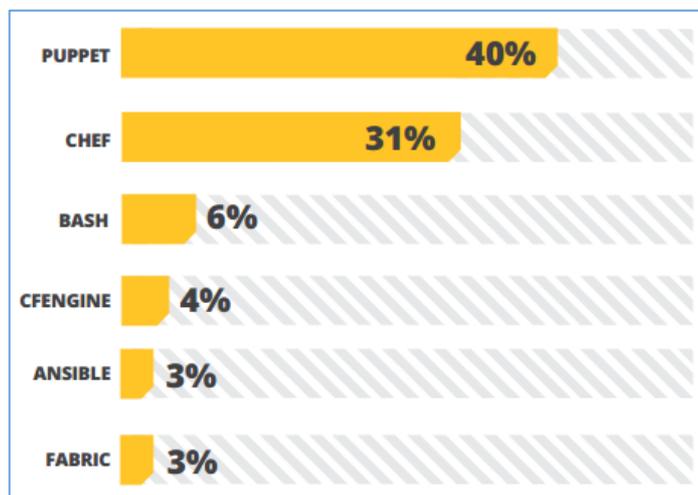
Fonte: White e outros (2013)

3.7 LEVANTAMENTO DAS CONFIGURAÇÕES DOS AMBIENTES PARA DEFINIÇÃO DE UM GERENCIADOR DE INFRAESTRUTURA

O provisionamento de infraestrutura deve ser um processo automatizado para que seja possível restabelecer, dentro de um período de tempo previsível, uma nova configuração programada em casos de falhas de hardware.

A escolha de uma ferramenta é essencial para reduzir o tempo desperdiçado com a configuração de ambientes. A Figura 7 mostra as ferramentas mais populares de configuração de infraestrutura.

FIGURA 7
Popularidade das ferramentas de configuração de infraestrutura



Fonte: White e outros (2013)

O Puppet (<http://puppetlabs.com/puppet/puppet-open-source>) é uma ferramenta que centraliza as configurações em um único ponto, permitindo que sejam distribuídas para outros nós dentro de uma rede. Utiliza linguagem declarativa para a configuração de sistemas operacionais, entre eles Linux, Windows e Solaris. Entre suas principais funcionalidades está a gerência de configuração do ambiente e a automatização da instalação de pacotes. Sua implantação permitiu maior controle sobre as configurações dos servidores, tornando possível sua reprodução para criação de ambientes de homologação.

3.8 ESTABELECIMENTO DE UMA SOLUÇÃO PARA A IMPLANTAÇÃO DA APLICAÇÃO EM AMBIENTE DE PRODUÇÃO

A implantação da aplicação em um ambiente de produção é o ponto crucial da Entrega Contínua, pois trata o ponto mais crítico que depende de todos os processos anteriores além de uma ferramenta para tornar a implantação automatizada.

Um *pipeline* de implantação no servidor de IC ajuda a remover ineficiências do processo, tornando o ciclo de *feedback* mais rápido e poderoso. Não existe uma

solução única para a criação de um *pipeline*, as configurações podem variar de acordo com a complexidade e arquitetura de cada projeto.

A equipe de desenvolvimento do “Novo Integra” optou por utilizar o Build Pipeline Plugin (<https://wiki.jenkins-ci.org/display/JENKINS/Build+Pipeline+Plugin>) no Jenkins. Através dele é possível interligar os *jobs* do Jenkins e definir se serão disparados de forma manual ou automática.

Sua implantação além de permitir uma visualização mais ampla do progresso dos processos da Entrega Contínua, também permitiu a identificação do uso de dependências desnecessárias em alguns componentes, serviços e módulos do sistema.

3.9 ANÁLISE DOS RESULTADOS OBTIDOS

Para identificar em que ponto uma organização está, em termos de maturidade de processos e práticas, Humble e Farley (2014) definiram um modelo de maturidade para a gerência de configuração e entrega de versão. O modelo considera as práticas de gestão de construção e IC, ambientes e implantação, gestão de entrega de versão e observância, testes e gerência de configuração. A progressão baseada em níveis pode ser utilizada para a melhoria contínua dos processos.

São apresentados cinco níveis globais, que podem ser utilizados para avaliação geral da organização. São eles:

- Nível 3 - Otimização: Foco em melhoria de processo
- Nível 2 - Gerido quantitativamente: Processos medidos e controlados
- Nível 1 - Consistente: Processos automatizados aplicados ao longo de todo o ciclo de vida da aplicação.
- Nível 0 - Repetível: Processo documentado e parcialmente automatizado
- Nível -1 - Regressivo: Processos que não podem ser repetidos, pouco controlados e reativos

O uso das ferramentas utilizadas durante a confecção desse artigo permitiu um crescimento que passou do nível regressivo para o repetível. A automatização dos processos reduziu o tempo desperdiçado com processos manuais, no entanto, ainda existem pontos que precisam ser melhorados para que o nível de otimização seja alcançado.

4 CONCLUSÃO

Entrega contínua é um assunto relativamente novo, no entanto, a partir do momento que se entende quais são as práticas envolvidas, é possível buscar por soluções específicas para cada ponto do processo.

A metodologia apresentada no capítulo 3 possibilitou a automatização dos principais processos da entrega de software e pode ser utilizada por qualquer equipe que pretenda implantar a Entrega Contínua para obter redução do tempo desperdiçado com tarefas manuais e aumentar a qualidade do software. A escolha das ferramentas pode variar de acordo com o escopo e a necessidade de cada projeto, cabe a cada equipe avaliar quais são as melhores soluções.

O estudo de caso permite concluir que a implantação da Entrega Contínua deve ser realizada de forma gradativa. Os níveis mais altos de maturidade poderão ser alcançados através da colaboração diária de todos os membros envolvidos na confecção do software. A prática diária levará a uma evolução natural com o decorrer do tempo, que permitirá o início de buscas por novos procedimentos a partir do momento que todos entenderem o real significado de entregar valor ao cliente.

A implantação é muito flexível, existem muitas opções de ferramentas para resolver um mesmo problema. O ponto positivo é que havendo alguma ferramenta que não atenda a alguma necessidade do processo, será possível buscar outra solução. O ponto negativo é decidir quais ferramentas usar no início da implantação. Esse foi o principal desafio encontrado durante a elaboração desse artigo.

As avaliações a respeito de trabalhos futuros mostram que o trabalho está longe de um fim. Muitos tópicos avançados podem ser explorados com o objetivo de alcançar melhores níveis de maturidade da Entrega Contínua. Entre eles está a gerência de dados, a implantação na nuvem, os sistemas avançados de monitoramento, os *pipelines* de entregas complexas e a orquestração de implantação. Além disso, algumas mudanças culturais são essenciais para ajudar na melhoria contínua dos processos. Uma das práticas que surgiu recentemente é denominada DevOps que sugere a colaboração entre os times de desenvolvimento e de operações. Com base nos resultados obtidos é possível vislumbrar uma vasta continuação para esse assunto ainda tão pouco explorado.

REFERÊNCIAS

BECK, Kent. **Extreme Programming Explained: Embrace Change**. Addison-Wesley Professional, 1999.

DANTAS, Cristine. **Gerência de Configuração de Software**: Desenvolvendo software de forma eficiente e disciplinada. Rio de Janeiro, 2009. Disponível em: <<http://www.devmedia.com.br/gerencia-de-configuracao-de-software/9145>>. Acesso em 20 jun 2014.

DIAS, Klessis Lopes; LUCENA, Carlos José Pereira de; KULESZA, Uirá. **Um framework orientado a aspectos para monitoramento e análise de processos de negócio**. 2008. 68f. Dissertação (Mestrado em Informática)-Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2008.

ESTUBLIER, Jacky et al. **Impact of the research community on the field of software configuration management**: summary of an impact project report. ACM SIGSOFT Software Engineering Notes, v. 27, n. 5, p. 31-39, 2002.

FOWLER, Martin. **Continuous Integration**. Disponível em: <<http://martinfowler.com/articles/continuousIntegration.html>>. Acesso em 7 fev 2014.

GARCIA, Francisco A. Integração Contínua: da teoria à prática – Parte 1. **Java Magazine**, Rio de Janeiro, n. 117, 2013. Disponível em: <<http://www.devmedia.com.br/integracao-continua-da-teoria-a-pratica-parte-1-revista-java-magazine-117/28284>>. Acesso em 15 abr 2014.

HUMBLE, Jez; FARLEY, David. **Entrega contínua**: como entregar software de forma rápida e confiável. Porto Alegre: Bookman, 2014.

LOPES, Juan. Integração Contínua. **.net Magazine**, Rio de Janeiro, n. 85, Porto Alegre: Bookman, 2011. Disponível em <<http://www.devmedia.com.br/integracao-continua-artigo-net-magazine-85/21086>>. Acesso em 15 abr 2014.

MURTA, Leonardo Gresta Paulino. **Gerência de Configuração no Desenvolvimento Baseado em Componentes**. Rio de Janeiro, 2006. Disponível em <<http://reuse.cos.ufrj.br/prometeus/publicacoes/odyssey-scm.pdf>>. Acessado em 27 jun 2014.

PRESSMAN, Roger S. **Engenharia de Software**. 5. Ed. São Paulo: Ed. McGraw-Hill, 2002.

SALES, Ernani O; LIMA REIS, Carla A; REIS, R. Q. **Apoio a Gerência de Configuração de Artefatos de Software integrado a Execução de Processos de Software**. Disponível em: <http://www3.ufpa.br/webapsee/index.php?option=com_docman&task=doc_view&gid=74&tmpl=component&format=raw&Itemid=43&lang=us>. Acesso em 20 mar 2014.

SATO, Danilo. **DevOps na prática**: entrega de software confiável e automatizada. São Paulo: Caso do Código, 2014.

SETE, Márcio. **Continuous Delivery**: Passos para implementação. **Engenharia de Software Magazine**, Rio de Janeiro, n. 65, 2013. Disponível em: <<http://www.devmedia.com.br/continuous-delivery-passos-para-implementacao/29783>>. Acesso em 19 fev 2014.

SHELAJEV, Oleg; MUUGA, Sigmar; MAPLE, Simon; WHITE, Oliver. **The Wise Developers' Guide to Static Code Analysis featuring FindBugs, Checkstyle, PMD, Coverity and SonarQube**. Disponível em: <<http://zeroturnaround.com/rebellabs/developers-guide-static-code-analysis-findbugs-checkstyle-pmd-coverity-sonarqube/>>. Acesso em 05 mai 2014.

TERRA, Ricardo; BIGONHA, Roberto S. **Ferramentas para Análise Estática de Códigos Java**. Belo Horizonte, 2008. Disponível em: <http://ebts2008.cesar.org.br/artigos/EBTS2008-Ferramentas_para_Analise_Estatica_de_Codigos_Java.pdf>. Acessado em 25 jun 2014.

WHITE, Oliver; BADRINARAYANAN, Krishnan; KABANOV, Jevgeni. **DevOps / ITops Productivity Report 2013**. Disponível em: <<http://zeroturnaround.com/rebellabs/rebel-labs-release-it-ops-devops-productivity-report-2013/>>. Acesso em 05 mai 2014.

WHITE, Oliver; MAPLE, Simon; MUUGA, Sigmar; TIMOŠIN, Juri; RASMUSSEN, Michael. **Java Build Tools – Part 2**: A Decision Maker's Comparison of Maven, Gradle and Ant + Ivy. Disponível em: <<http://zeroturnaround.com/rebellabs/java-build-tools-part-2-a-decision-makers-comparison-of-maven-gradle-and-ant-ivy/>>. Acesso em 05 mai 2014.