Implementação de um Fluxo de Integração Contínua utilizando Hudson em um Ambiente de Desenvolvimento Distribuído

Breno Carvalho Costa[1], Thales Vieira Batista[1], Evandrino Barros[1], Celio Trois[2]

[1]Centro Federal de Educação Tecnológica de Minas Gerais (CEFET-MG). e-mails: breno@lsi.cefetmg.br, thalesvb@dri.cefetmg.br, ebarros@decom.cefetmg.br
[2]Colégio Técnico Industrial de Santa Maria (CTISM). e-mail: celio.trois@gmail.com

Resumo: Projetos de *software* de grande porte frequentemente passam por dificuldades na integração de seus componentes. Para superar isto, a integração contínua (IC) propõe a integração em incrementos menores, para diminuir os riscos e aumentar a qualidade. Neste contexto, foi proposta a aplicação de IC no projeto SIGA-EPCT – sistema de gestão acadêmica, desenvolvido de forma colaborativa por grupos distribuídos geograficamente – utilizando o servidor Hudson. Para tanto, foi criado um fluxo de IC com os objetivos de integrar o *software* a cada nova revisão, diminuindo o tempo no tratamento de eventuais falhas de construção e criar um mecanismo de aprendizagem com os *bugs* de codificação encontrados. Como resultado, observou-se que falhas na construção são apontadas imediatamente aos responsáveis, que as resolvem, mantendo o *software* estável. O envio de um relatório aos desenvolvedores possibilita que eles aprendam com os *bugs* encontrados no código fonte, tornando não somente o *software* maduro, mas também os desenvolvedores mais capacitados.

Palavras–chave: integração contínua, Hudson, desenvolvimento distribuído

1. INTRODUÇÃO

Projetos de *software* de grande porte frequentemente passam por dificuldades na integração de seus componentes, sobretudo, quando a integração é feita ao fim de várias iterações do desenvolvimento. Consequentemente, isso pode trazer problemas de qualidade do produto final. Neste contexto, a integração contínua (IC) propõe a construção em incrementos menores, para diminuir os riscos e aumentar a qualidade de um projeto. Com isso, pretende-se detectar erros o mais breve possível e fazer as correções necessárias.

O projeto SIGA-EPCT é um sistema de gestão acadêmica, que está sendo desenvolvido de forma colaborativa por instituições federais, apoiados pelo Ministério da Educação (SIGA-EPCT, 2012). Este projeto possui um modelo de gestão colaborativa composto por grupos de desenvolvimento distribuídos geograficamente. Um grande número de desenvolvedores realizam alterações no código fonte do projeto e, com isso, é necessário tratar erros encontrados o mais rápido possível para manter a estabilidade e qualidade. Neste sentido, um fluxo de IC foi criado para atender as necessidades do SIGA-EPCT e o servidor Hudson foi adotado para implementar este fluxo. Os objetivos desta implementação foram: (1) integrar o *software* a cada nova revisão, relatando as informações da integração aos usuários responsáveis e diminuindo o tempo no tratamento de eventuais falhas de construção; (2) criar um mecanismo de aprendizagem sobre os *bugs* de codificação para melhorar a qualidade do código fonte ao longo do tempo.

O trabalho está organizado como se segue. Na Seção 2, conceitos sobre IC são apresentados e o servidor Hudson é descrito. A Seção 3 descreve a aplicação de IC ao projeto SIGA-EPCT e os resultados são apresentados na Seção 4. Por fim, a Seção 5 apresenta as conclusões do trabalho.

2. INTEGRAÇÃO CONTÍNUA

O processo de integrar *software* não é um problema novo, talvez não seja um problema para projetos pequenos com poucas dependências externas, mas a complexidade de um projeto aumenta quando existe maior necessidade de integrar seus componentes para trabalharem juntos. Esperar até o

ISBN 978-85-62830-10-5

fim do ciclo de um projeto para integrá-lo pode trazer vários tipos de problemas de qualidade, que frequentemente são a causa de atrasos no cronograma. A IC trata esses riscos mais rapidamente, realizando a integração em incrementos pequenos.

Segundo Fowler (2000), IC é "uma prática de desenvolvimento de *software* onde membros de uma equipe integram seus trabalhos frequentemente (geralmente uma vez por dia). Cada integração é verificada por um *script* automatizado para detectar erros o mais rapidamente possível. Muitas equipes verificam que essa abordagem leva a reduzir significantemente os problemas de integração e permitem que uma equipe desenvolva *software* coesivo mais rapidamente".

Observando essas características, Duvall, Matyas e Glover (2007) propuseram um cenário para implementar IC (Figura 1).

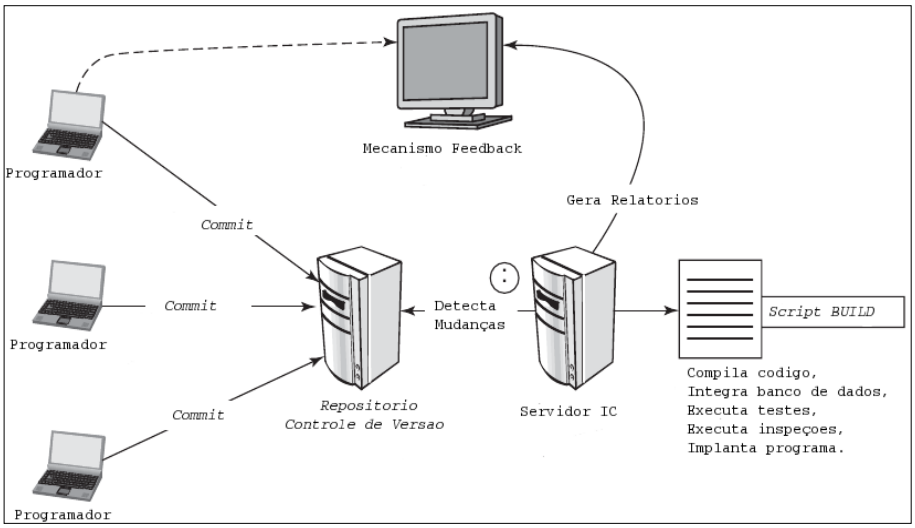

Figura 1: Componentes de um sistema de IC. Adaptado de Duvall, Matyas e Glover (2007)

Neste cenário, programadores iniciam a integração (outros membros da equipe podem participar desta etapa, como por exemplo, os administradores de banco de dados) fazendo *commit* dos arquivos modificados localmente para o servidor de controle de versões. A partir disso, o servidor IC detecta mudança na revisão e inicia o processo de integração, ou seja, ele faz o *checkout* com as modificações no projeto e executa o *script* que automatiza a construção. Neste *script*, pode-se ter configurações diferentes para cada implementação, mas é essencial que ele compile o projeto e execute os testes unitários e funcionais. Depois disso, o servidor IC gera relatórios apontando os erros e sucessos da integração e repassa aos responsáveis usando um mecanismo de *feedback*.

Segundo Duvall, Matyas e Glover (2007), algumas características da IC são importantes:

- Todos os desenvolvedores executam construções locais em suas próprias estações de trabalho antes de submeter os arquivos de código fonte ao repositório de controle de versões para verificar que as mudanças não quebrarão a construção;
- Desenvolvedores submetem os códigos ao repositório de controle de versões pelo menos uma vez ao dia;
- A integração é executada em uma máquina de construção separada;
- Todos os testes devem passar em cada construção e um artefato – arquivo executável – é gerado ao fim dela;
- Consertar construções quebradas é a prioridade máxima;
- Alguns desenvolvedores revisam relatórios gerados pela construção, tal como padrões de codificação e relatórios de análises de dependências, para buscar áreas de melhoria.

Para viabilizar o cenário proposto, é necessário instalar e configurar um servidor de IC. Para isso, existem vários servidores com propostas semelhantes, que variam nos meios de instalação, configuração e usabilidade. Dentre os servidores de IC mais conhecidos, pode-se citar:
Hudson[1], Jenkins[2], Damage Control[3], Cruise Control[4], Janky[5], Bamboo[6], Continuum[7], AnthillPro[8]. Recentemente, o Hudson tem sido adotado por vários projetos ao redor do mundo devido às suas vantagens, tornando-se bastante popular (HUDSON, 2012).

2.1. HUDSON

Hudson é uma ferramenta de IC escrita em Java, que é executada em um servidor de aplicação, como Apache Tomcat ou GlassFish. Ele suporta ferramentas de gerência de configuração de *software* (SCM) incluindo CVS, Subversion, Git e pode executar projetos baseados em Apache Ant e Apache Maven, além de *scripts shell* ou comandos *batch* do Windows (ECLIPSE, 2012).

O Hudson monitora execuções de tarefas repetidas e foca em duas propostas principais (HUDSON, 2012):

1. Construir/testar projetos de *software* continuamente: Hudson provê um sistema de IC de boa usabilidade, facilitando para os desenvolvedores integrar mudanças ao projeto e para usuários obter uma nova construção;

2. Monitoração de execuções de tarefas externas à construção: é possível configurar *plugins* de envio de e-mail para os usuários receberem o *feedback* das construções, noticiando por eventuais falhas que possam ocorrer. Além disso, o Hudson guarda os *logs* de construção para facilitar visualizações posteriores.

Destaca-se algumas características principais: facilidade na instalação e configuração, suporte à lista de mudanças por revisão, integração com RSS, e-mail, mensagem instantânea, *tag* de artefatos, relatórios de testes do JUnit/TestNG, construções distribuídas, suporte para instalação de *plugins*. Ademais, outras características podem ser apontadas: interface gráfica de fácil configuração (dividida em duas partes, uma parte é dedicada ao administrador e outra para o usuário), licença livre, código aberto, larga comunidade de desenvolvedores e usuários, suporte do sistema feito pela Oracle, listas de discussões, vários *plugins*, segurança (controle de acesso por autenticação de usuário), fácil empacotamento de arquivos do diretório de trabalho e desenvolvimento de novos *plugins*. Devido a essas características, o Hudson tem sido adotado em anos recentes para implementação de IC em vários projetos ao redor do mundo e, por esta razão decidiu-se adotar o Hudson para implementar IC no projeto SIGA-EPCT.

3. INTEGRAÇÃO CONTÍNUA NO SIGA-EPCT

O SIGA-EPCT é um sistema de gestão acadêmica, que está sendo desenvolvido com tecnologias livres e de forma colaborativa pelas próprias instituições participantes da Rede de Educação Profissional, Científica e Tecnológica. Este projeto tem o apoio do Ministério da Educação, por meio da Secretaria da Educação Profissional e Tecnológica - SETEC (SIGA-EPCT, 2012).

O objetivo deste projeto é planejar, desenvolver, implementar e dar suporte ao Sistema Integrado de Gestão Acadêmica - SIGA-EPCT, provendo às unidades acadêmicas supervisionadas pela SETEC/MEC de instrumentos e ferramentas que contribuam para sua gestão efetiva, tanto acadêmica quanto administrativa.

O SIGA-EPCT possui um modelo de gestão colaborativa composto de um gerente de projeto, cinco coordenadores de área e grupos que executam atividades específicas, permanecendo ainda a estrutura de núcleo composto por gestor, orientador e desenvolvedor. Os papéis definidos para a gestão colaborativa são: Gerência de Projeto, Coordenação de Requisitos, Coordenação de Modelagem e Banco de Dados, Coordenação de Produto e Integração, Coordenação de Codificação e Tecnologia, Coordenação de Infraestrutura e Coordenação do Processo de Desenvolvimento.

---

[1] Disponível em: <http://hudson-ci.org/>

[2] Disponível em: <http://jenkins-ci.org/>

[3] Disponível em: <http://damagecontrol.codehaus.org/>

[4] Disponível em: <http://cruisecontrol.sourceforge.net/>

[5] Disponível em: <http://github.com/github/janky/>

[6] Disponível em: <http://www.atlassian.com/software/bamboo/>

ISBN 978-85-62830-10-5

[7] Disponível em: <http://continuum.apache.org/>
[8] Disponível em: <http://www.urbancode.com/html/products/anthillpro/default.html/>

A equipe de desenvolvimento é formada por núcleos de pesquisa e desenvolvimento instalados em instituições da rede federal de educação profissional e tecnológica nas diversas regiões do país. O SIGA-EPCT possui os seguintes núcleos: IF Farroupilha, IF da Bahia, IF de Alagoas, IF do Tocantins, IF Fluminense, CEFET-MG, IF de Santa Catarina, CTISM, IF do Mato Grosso e IF Catarinense.

O fato do SIGA-EPCT ser de código aberto e ter vários desenvolvedores em seu projeto distribuídos geograficamente, implica na necessária integração de componentes de *software* em espaços curtos de tempo, pois os erros são tratados rapidamente, as falhas sanadas e o produto liberado com maior qualidade (DESHPANDE, A.; RIEHLE, D., 2008). Neste contexto, optou-se por implementar a IC, escolhendo-se o Hudson como servidor, em virtude das características descritas na Seção 2.1.

3.1. FLUXO APLICADO AO SIGA-EPCT

O cenário proposto por Duvall, Matyas e Glover (2007) foi adaptado para a realidade do SIGA-EPCT, de modo que atenda as necessidades do projeto. Para tanto, buscou-se atender dois aspectos principais: integrar o *software* a cada nova revisão, relatando as informações da integração aos usuários responsáveis e criar um mecanismo de aprendizagem sobre os *bugs* de codificação. O fluxo da Figura 2 mostra como a IC foi aplicada.

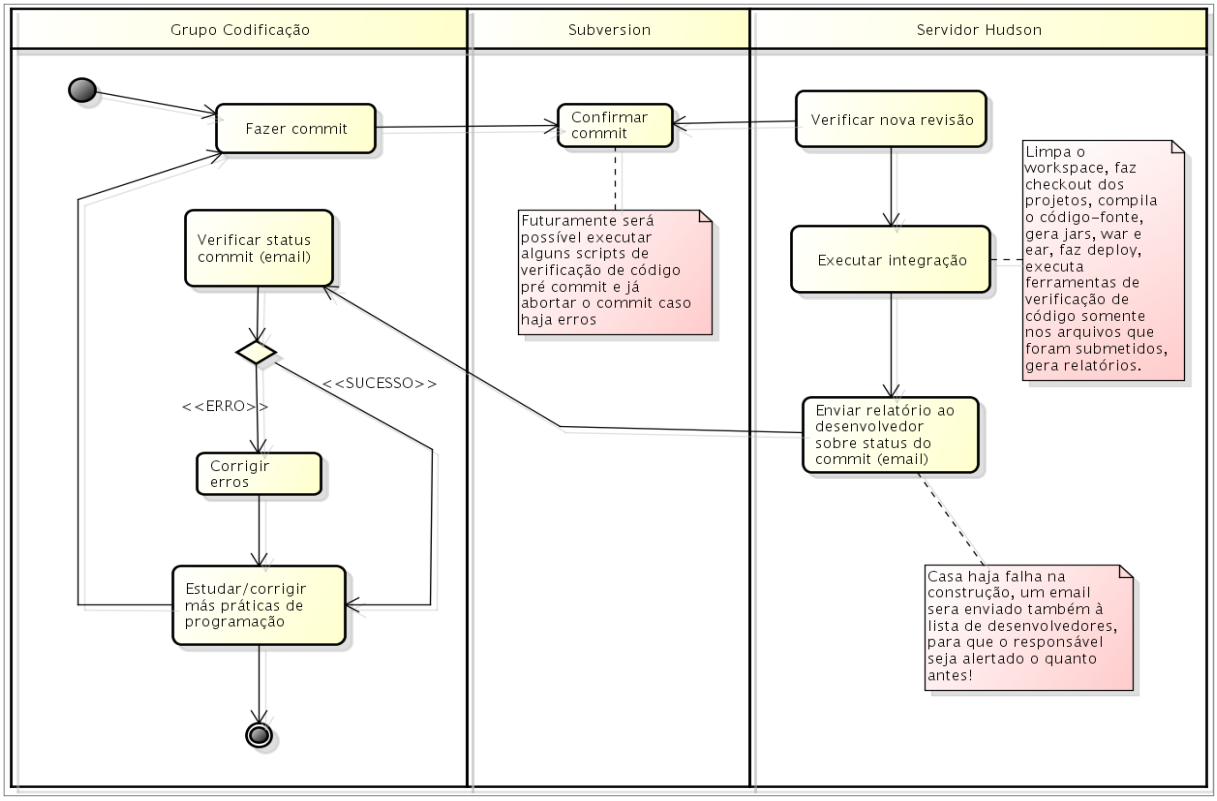

Figura 2: Fluxo de IC aplicado ao SIGA-EPCT

O fluxo contém oito etapas:

1. Fazer *commit*: um programador do grupo de codificação faz o *commit* quando termina uma funcionalidade ou quando um *bug* é corrigido;
2. Confirmar *commit*: a nova revisão submetida pelo grupo de codificação é verificada pelo repositório de controle de versões;
3. Verificar nova revisão: o Hudson detecta mudanças no repositório de controle de versões e inicia uma nova integração;
4. Executar integração: o Hudson faz o *checkout* do projeto modificado, compila o código

fonte, gera os executáveis (jar, war e ear), implanta o *software* no servidor GlassFish e, por fim, executa ferramentas de verificação de código fonte nos arquivos que foram submetidos;

5. Enviar relatório ao desenvolvedor sobre status do commit (e-mail): quando há sucesso na integração, um e-mail é enviado ao programador que fez o *commit*. Mas caso haja falha na integração, um e-mail também é enviado à lista de desenvolvedores para que uma medida rápida seja tomada;
6. Verificar status do *commit*: o programador deve verificar o retorno da integração e caso haja erros deve corrigi-los, ou ainda, caso haja sucesso verificar suas más práticas de programação;
7. Corrigir erros: caso haja erros na construção, o programador deve consertá-los;
8. Estudar/corrigir más práticas de programação: alguns relatórios produzidos pelos *plugins* de verificação de código fonte devem ser verificados.

Para implementar este fluxo, adotou-se o servidor Hudson, devido às vantagens descritas na Seção 2.1. A próxima seção descreve sua instalação e configuração.

3.2. INSTALAÇÃO E CONFIGURAÇÃO DO SERVIDOR DE INTEGRAÇÃO

O Hudson foi instalado em um servidor GlassFish[1], em seguida, foi criada uma tarefa com o nome SIGAEPCT-IC-Branch<versão>, que faz a integração dos códigos fonte em desenvolvimento no projeto, de acordo com a versão especificada. Embora esta seja a tarefa principal utilizada nesta proposta, é possível configurar outras tarefas para fazer a integração em diferentes partes do repositório, de acordo com a necessidade. A Figura 3 mostra a tela principal do Hudson com algumas tarefas configuradas.

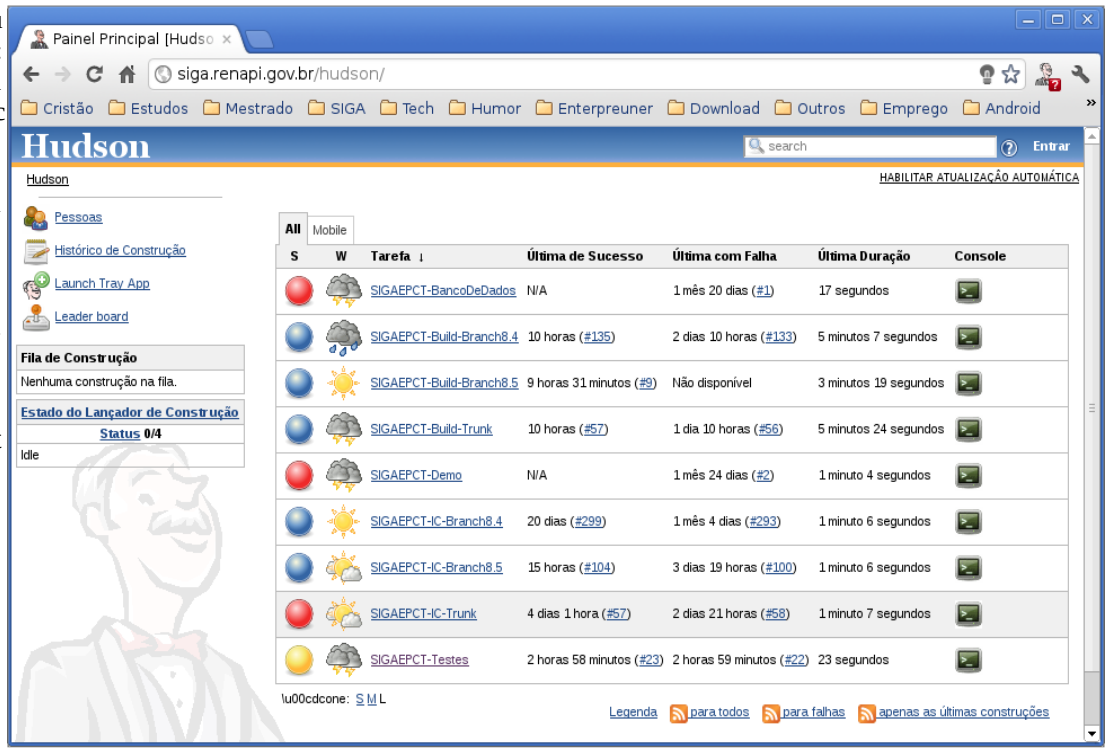

Figura 3: Tela principal do Hudson

O servidor de controle de versões do projeto SIGA-EPCT é o Subversion, então o *plugin Subversion*[2] foi configurado para realizar as tarefas SCM, ou seja, verificar mudanças no repositório e fazer

ISBN 978-85-62830-10-5

[1] Disponível em: <http://java.net/downloads/hudson/>
[2] Disponível em: <http://wiki.hudson-ci.org/display/HUDSON/Subversion+Plugin>
checkout do projeto. Quando a mudança na revisão do projeto é detectada, a compilação do projeto é feita pelo Apache Ant, que segue as etapas descritas no arquivo *build.xml* do projeto. Após a compilação, quatro *plugins* são executados para verificar o código fonte modificado em busca de más práticas de programação: *Findbugs*[1], *PMD*[2], *DRY*[3] e *Checkstyle*[4]. Por fim, o *plugin* para envio de e-mails[5] foi configurado para enviar as notificação aos responsáveis pela integração.

4. RESULTADOS

Para medir o impacto da implementação de IC no projeto SIGA-EPCT, dois experimentos foram realizados. No primeiro experimento, verificou-se o número de construções com sucesso e falha desde a versão 8.1 até a versão 8.4, o que permite comparar o número de construções com sucesso e falha e, com isso, verificar a vantagem do aumento no tempo de resposta em caso de falha na construção. No segundo experimento, as ferramentas de verificação estática de código fonte foram aplicadas na versão atual do projeto, procurando *bugs* no código fonte para que auxiliem os programadores a aprenderem com seus próprios erros, melhorando a qualidade do *software*. A partir disso, é possível verificar as vantagens da abordagem proposta.

4.1. CONSTRUÇÃO DO PROJETO

Uma das fases da integração do projeto é a construção, que utiliza um *script* automatizado para compilar o projeto. A partir do retorno da construção, é possível tomar providências caso haja alguma falha na construção, visando manter a estabilidade do projeto. A Figura 4 mostra o comportamento das construções nas últimas versões do projeto.

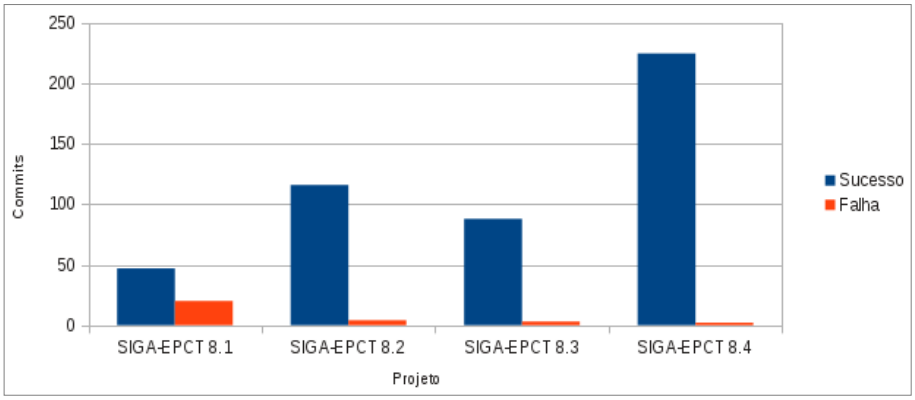

Figura 4: Comportamento das construções das últimas versões do projeto SIGA-EPCT

No projeto SIGA-EPCT 8.1, 47 construções foram feitas com sucesso e 20 com falhas, na versão 8.2 foram 116 construções com sucesso e 4 com falhas, na versão 8.3 foram 88 construções com sucesso e 3 com falhas e, por fim, a versão 8.4 apresentou 225 construções com sucesso e 2 com falhas. Nas versões anteriores (versão 8.4 e anteriores), quando havia alguma falha na construção do projeto nenhum aviso era direcionado aos responsáveis. Com a implementação de IC neste projeto, a partir de agora (versão 8.5) quando há alguma falha na construção, os responsáveis são acionados para consertar os erros e a estabilidade do projeto é mantida.

4.2. MECANISMO DE APRENDIZAGEM PELOS BUGS DE CODIFICAÇÃO

A utilização de ferramentas para verificar estaticamente o código fonte é muito importante para melhorar a qualidade do *software*, já que informam aos desenvolvedores sobre *bugs* frequentes de estilo, desempenho, dentre outros. Os *plugins* utilizados nesta implementação foram: *Findbugs, PMD, DRY e Checkstyle*.

---

[1] Disponível em: <http://wiki.hudson-ci.org/display/HUDSON/FindBugs+Plugin >

[2] Disponível em: <http://wiki.hudson-ci.org/display/HUDSON/PMD+Plugin>
[3] Disponível em: <http://wiki.hudson-ci.org/display/HUDSON/DRY+Plugin>
[4] Disponível em: <http://wiki.hudson-ci.org/display/HUDSON/Checkstyle+Plugin>
[5] Disponível em: <http://wiki.hudson-ci.org/display/HUDSON/Email-ext+plugin>

O Findbugs reconhece mais de 300 erros de programação e códigos duvidosos que podem ser identificados usando simples técnicas de análise. Além disso, também usa técnicas de análise mais sofisticadas, garantindo o desenvolvimento correto (AYEWAH N. et. al, 2008). Algumas categorias de *bug* encontrados pelo Findbugs são mais importantes:

- Más práticas: código que viola alguma boa prática de programação. Por exemplo, sobrescrever o método *equals()* e não sobrescrever o método *hashCode()*, comparar *string* com *equals()*;
- Correção: código incorreto que executa algo que o desenvolvedor não pretendia. Por exemplo, *loop* recursivo infinito;
- Vulnerabilidade de código malicioso: código apresenta vulnerabilidade para outro código malicioso. Por exemplo, uma variável estática deveria possuir o modificador *final*;
- Desempenho: código com desempenho ruim. Por exemplo, um método privado nunca é chamado, concatenação de *string* usando '+', usar construtor em vez do método *valueOf()*.

O PMD é semelhante ao Findbugs, pois busca encontrar problemas de codificação, mas varia as categorias de problemas encontrados. Alguns problemas podem ser destacados: não fechamento de chaves em métodos, exceção lançada sem nenhum tratamento feito, alta complexidade ciclomática, código fonte não usado.

O DRY significa *don't repeat yourself* e tem como objetivo verificar trechos de código fonte duplicados, que é uma prática errada de programação e traz dificuldades na manutenção do código.

O Checkstyle verifica problemas no estilo de codificação, como o não encapsulamento de campos, nomeação de atributos, métodos ou classes fora da convenção, importação não utilizada.

O *feedback* do Hudson aos usuários responsáveis com as informações acima pretende fazer com que os programadores corrijam os *bugs* encontrados no código fonte, aprendam com as informações geradas e não os cometam mais. A verificação de *bugs* na versão atual do projeto SIGA-EPCT (versão 8.5) é apresentada na Figura 4.

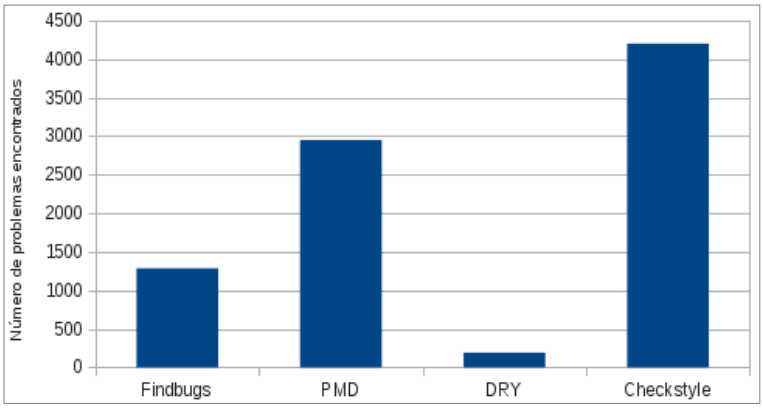

Figura 4: Problemas encontrados no SIGA-EPCT

Nota-se que o Findbugs encontrou 1285 *bugs*, o PMD encontrou 2950, o DRY encontrou 191 e o Checkstyle encontrou 4200. Com a implementação de IC neste projeto, os problemas encontrados no código fonte serão corrigidos pelos responsáveis, uma vez que estes recebem o relatório de problemas detectados pelo Hudson. Pretende-se que, ao longo do tempo, todos os problemas sejam sanados rapidamente e a qualidade do projeto aumente consideravelmente.

ISBN 978-85-62830-10-5

5. CONCLUSÕES

A implementação de IC utilizando o Hudson no projeto SIGA-EPCT foi realizada para contemplar dois objetivos: (1) integrar o *software* a cada nova revisão, para diminuir o tempo de resposta em caso de eventuais falhas e manter a estabilidade do projeto; (2) criar um mecanismo de aprendizagem sobre os *bugs* de codificação.

Conclui-se que a construção contínua é importante para diminuir o tempo de resposta no tratamento de falhas do sistema. Isto mantém o *software* estável e consistente ao longo do tempo e não somente no lançamento de uma nova versão. Espera-se com isso que aumente a qualidade do SIGA-EPCT, já que agora as construções são feitas em pequenos incrementos, além de diminuir o tempo de tratamento de eventuais falhas na construção. Ademais, é possível obter um arquivo executável do sistema a cada nova revisão do projeto, de modo que se possa ter um servidor de demonstração do aplicativo sempre atualizado.

Desde o início do desenvolvimento do SIGA-EPCT, as más práticas de programação ainda não haviam sido verificadas. Atualmente na versão 8.5, cogitou-se que o número de problemas encontrados seria alto. Para medir isso, foi feita a verificação estática de código fonte, por meio de *plugins* instalados no Hudson. O número de problemas encontrados foi relativamente alto (8626 no total). Com o objetivo de melhorar a qualidade dos códigos criados pelos programadores, a implementação de um mecanismo de *feedback* permite enviar relatórios de *bugs* encontrados no código fonte enviado pelo programador. A partir disso, o programador pode estudar suas más práticas de programação, melhorando sua capacidade de desenvolvimento e a maturidade do projeto desenvolvido.

Algumas propostas futuras podem complementar a metodologia aplicada neste trabalho. Pode-se criar um *script* pré-*commit* para ser executado antes que o Subversion aceite um *commit* feito por um programador. Neste *script*, um *commit* poderia ser interrompido caso alguma falha na construção do projeto fosse encontrada. Além disso, outros *plugins* de verificação de código fonte poderiam ser instalados para melhorar as informações retornadas ao usuário. Para calcular as métricas do projeto, poderiam ser instalados *plugins* específicos para tal função, tais como JavaNCSS ou Sonar. Outra função que pode ser implantada é distribuir o executor de tarefas do Hudson em várias computadores, utilizando conceitos de sistemas distribuídos.

REFERÊNCIAS

AYEWAH, N.; HOVEMEYER, D.; MORGENTHALER, J. D.; PENIX, J.; PUGH, W. Using Static Analysis to Find Bugs. **IEEE Software**, v. 25, n. 5, set 2008.

DESHPANDE, A.; RIEHLE, D. **Continuous Integration in Open Source Software Development**. In: Proceedings of the Fourth Conference on Open Source Systems (OSS 2008). Springer Verlag, 2008. Page 273-280.

DUVALL, P. M.; MATYAS, S.; GLOVER, A. **Continuous Integration: Improving Software Quality and Reducing Risk**. 1. ed. Addison-Wesley, 2007.

ECLIPSE Foundation. **Hudson – Continuous Integration Server**. Disponível em: <http://www.eclipse.org/hudson>. Acesso em: 29 jun. 2012.

FOWLER, M. **Continuous Integration**. Disponível em: <http://http://martinfowler.com/articles/continuousIntegration.html>. Acesso em: 29 jun. 2012.

HUDSON. **Hudson-ci/Meet Hudson – Getting Started With Hudson**. Disponível em: <http://wiki.eclipse.org/Hudson-ci/Meet_Hudson>. Acesso em: 30 jun. 2012.

JEZ, H; FARLEY, D. **Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation**. 1 ed. Addison-Wesley. 2011. ISBN 978-0-321-60191-9.

SIGA-EPCT. **Conheça o projeto**. Disponível em: <*http://www.renapi.gov.br/sigaepct/o-projeto*>. Acesso em: 25 jun. 2012.

SHORE, J. Continuous Design. **IEEE Software**, Portland, v. 21, n.1, jan-fev 2004.

ISBN 978-85-62830-10-5