Integração Contínua com Software Livre: Um relato de implantação na Fundação de Amparo a Pesquisa do Estado de Alagoas – FAPEAL

Felipe B. de Queiroz[1], Leonardo F. M. de Oliveira[1]

[1]Fundação de Amparo a Pesquisa do Estado de Alagoas - FAPEAL
CEP 57.020-330 – Maceió – AL – Brasil

{felipe.buarque, leonardo.fernandes}@fapeal.br

ABSTRACT. *The task of integrating artifacts is inherent in the process of software development, where all parts are being worked by different team members are put together and a check is made whether the system is functioning according to requirements. Therefore, Continuous Integration emerged as a strategy to attach the artifacts used in the software development process constantly and in short cycles. This work aims to demonstrate the process of adopting the IC practice using open source software in the Fundação de Amparo a Pesquisa do Estado de Alagoas - FAPEAL, approaching its concepts and demonstrating the benefits of their use.*

RESUMO. *A tarefa de integrar artefatos é inerente ao processo de desenvolvimento de software, onde todas as partes que estão sendo trabalhadas pelos diferentes membros da equipe são colocadas juntas e é feita uma verificação se o sistema está funcional, conforme os requisitos. Visto isso, a Integração Contínua surge como uma estratégia de unir os artefatos usados no processo de desenvolvimento de software de maneira constante e em ciclos curtos. Assim, este trabalho tem por objetivo demonstrar o processo de adoção da prática de IC utilizando software livre na Fundação de Amparo a Pesquisa do Estado de Alagoas - FAPEAL, abordando seus conceitos e demonstrando os benefícios de sua utilização.*

1. INTRODUÇÃO

O processo de integrar software não é um problema novo [Duval et al. 2007]. No processo de desenvolvimento de software, toda equipe, com duas ou mais pessoas, precisa lidar com a tarefa de integrar seus artefatos. À medida que a complexidade do projeto aumenta, há uma grande necessidade de tais artefatos serem integrados com uma certa frequência, afim de manter os mesmos atualizados em relação ao projeto e assegurar que os componentes trabalhem juntos e de maneira funcional. Visto isso, a falta de automação e adiamento para o final do ciclo de desenvolvimento o deixa passivo de alguns problemas, como o esquecimento por parte do responsável pela integração da inclusão de algum artefato, ou descobrir que dois desenvolvedores usaram versões diferentes de bibliotecas de terceiros o que acarretou em erro no momento da integração.

Assim, a Integração Contínua (IC) surge como uma estratégia de unir os artefatos utilizados no processo de desenvolvimento de software de maneira constante e em ciclos curtos, permitindo aos desenvolvedores a habilidade de realizar alterações no código, sabendo que,

se o sistema "quebrar", os mesmos possam ser alertados imediatamente e tenham mais tempo para corrigir o problema e se ajustar às mudanças mais rapidamente.

Paralelo a isso, o alto grau de maturidade alcançado pelas ferramentas livres nos últimos anos, principalmente aquelas que auxiliam os ciclos de desenvolvimento de software, vem permitindo sua adoção nos mais diferentes contextos, inclusive servindo como base para a estratégia de IC.

Portanto, este trabalho relata o processo de adoção da prática de IC utilizando software livre pelo time de desenvolvimento da Fundação de Amparo a Pesquisa do Estado de Alagoas - FAPEAL, procurando abordar seus conceitos e apresentar, qualitativamente, os benefícios de sua utilização, objetivando contribuir com estudos empíricos relacionados à produtividade de equipes de desenvolvimento de software.

2. INTEGRAÇÃO CONTÍNUA

A Integração Contínua é definida como um processo que realiza as tarefas de construção, teste, análise e *deploy* de uma aplicação, para assegurar que a mesma funcione corretamente e siga as melhores práticas [Kawalerowics and Berntson 2010]. Todo esse processo é executado a cada modificação no código fonte e provê um *feedback* imediato ao time de desenvolvimento, como será descrito em mais detalhes na seção 4.

Reunindo o que foi escrito por outros autores ([Fowler 2006], [Duval et al. 2007] e [Kawalerowics and Berntson 2010]), esta definição enfatiza que o processo de IC não se restringe apenas a compilar sua aplicação, mas sim, junto a isso, rodar os testes, os analizadores e disponibilizar o software, de maneira que todo este processo seja automatizado e que ao seu final, um mecanismo de *feedback* informe à equipe o status do projeto. É por isso que muitos autores afirmam que a IC é o "gatilho" para outras definições, como *Continuous Compilation* [Saff and Ernst 2004], *Continuous Building* [Kim et al. 2008], *Continuous Testing* [Saff and Ernst 2004], *Continuous Deployment* [Duval et al. 2007], dentre outras.

De acordo com [Duval et al. 2007], num nível mais alto, a adoção de IC numa equipe de desenvolvimento traz os seguintes valores:

- **Redução de riscos:** Dado que a integração, execução dos testes e inspeções são feitas várias vezes ao dia, existe grande chance de descobrir um erro assim que o mesmo é introduzido;
- **Redução de processos manuais repetitivos:** Atividades como, compilação de código, integração de banco de dados, testar, inspecionar, disponibilizar e *feedback* são encorajadas a serem automatizadas o que torna cada tarefa muito menos propensa a erros;
- **Geração de software "implantável" a qualquer momento:** Dado que o processo roda da mesma maneira, em todo momento, ao final do ciclo da IC é gerado um software funcional. O que ocorre a cada pequena mudança em algum artefato;
- **Permite melhor visibilidade do projeto:** A redução de suposições sobre o andamento do projeto ocorre de maneira natural, visto que a IC provê informações a todo momento sobre recentes construções, métricas de qualidade entre outras;
- **Estabelece maior confiança da equipe no produto:** Em toda construção, a equipe saberá que testes estarão rodando sobre o software e analisadores verificarão aderência aos padrões estabelecidos, o que encoraja o time a realizar alterações sempre que necessárias.

Levando em consideração tais valores, observa-se que a adoção do processo de IC complementa outras práticas do desenvolvimento de software, como a aderência aos padrões de código, *refactoring* e *releases* curtos, não importando se o projeto utiliza RUP [Sommerville 2007], XP [Teles 2004], SCRUM [Schwaber 2001], Crystal [Cockburn 2009], ou qualquer outra metodologia.

3. MÉTODO DE PESQUISA

Este trabalho utilizou o método de pesquisa-ação, o qual é um método intervencionista, que permite ao pesquisador testar hipóteses sobre o fenômeno de interesse implementando e acessando as mudanças no cenário real [Lindgren et al. 2004].

Conforme [Stringer 1996], a pesquisa-ação compreende uma rotina composta por três ações principais: observar, para reunir informações e construir um cenário; pensar, para explorar, analisar e interpretar os fatos; e agir, implementando e avaliando as ações. Tais ações se refletem no contexto da FAPEAL, onde se fez necessário observar o ambiente e perceber os problemas reais relacionados ao desenvolvimento de software, de modo a buscar e implantar soluções, com o foco essencial de investigar a adoção da prática de IC utilizando ferramentas livres pela equipe de desenvolvimento da Fundação.

A escolha dos sujeitos se fez de maneira intencional, estando constituída de 5 pessoas, sendo 4 (quatro) desenvolvedores e 1 (um) designer, lotadas na Unidade Gestora de Tecnologia da Informação (UGTI) da FAPEAL, representando a equipe que utilizou as práticas de IC no desenvolvimento de 1 (um) projeto. Assim, os dados primários da pesquisa foram coletados por meio de exame dos artefatos do projeto, que é um método no qual o pesquisador examina documentos, materiais e/ou artefatos, e registra os dados sobre instrumentos previamente preparados. Por meio de exame dos artefatos do projeto, pôde-se constatar a melhoria ou não, na qualidade do produto resultante e na velocidade de desenvolvimento imposta no projeto para alcançar o produto final.

4. IMPLANTAÇÃO

Durante um período de 04 meses (Outubro de 2009 a Janeiro de 2010), buscou-se implantar algumas técnicas de IC visando prover maior velocidade à equipe de desenvolvimento e maior qualidade nos produtos apresentados, utilizando como base o projeto de um sistema para controle financeiro da Fundação. Os membros da equipe de desenvolvimento já adotavam práticas de algumas metodologias de desenvolvimento (Scrum e XP) como código coletivo, programação em par, reuniões diárias, *releases* curtas (02 semanas) etc., e procuraram adaptar-se para a adoção de mais uma prática no dia-a-dia.

4.1. Ambiente de Desenvolvimento

Algumas ferramentas são essenciais para adoção da prática de IC. Seguindo o modelo de migração para software livre adotado pela FAPEAL no início de 2009, procurou-se configurar um ambiente 100% livre no que diz respeito às ferramentas utilizadas pelos desenvolvedores. Tal modelo foi proposto, a princípio, com o objetivo de reduzir ou, se possível, eliminar os custos com licenças de software, os quais eram responsáveis por cerca de 6, 5% do orçamento anual da UGTI. A descrição das principais ferramentas utilizadas no processo de IC segue abaixo.

- **Sistema de controle de versão (SCV):** Seu propósito é gerenciar mudanças no código fonte e outros artefatos (como documentação), provendo um ponto de acesso único de acesso controlado ao repositório. Como se tratava de uma equipe de desenvolvimento pequena e com experiência prévia na ferramenta, o SCV escolhido foi o Subversion (SVN) [Collins-Sussman et al. 2008].
- **Servidor de IC:** Funciona como um gerenciador de atividades no processo de IC. Começando pela verificação de mudanças no repositório de controle de versão e recuperação dos artefatos, o servidor dispara *scripts*, como um *script* de Construção, e tarefas numa ordem preestabelecida. Devido a facilidade de configuração, a ferramenta escolhida para tal tarefa foi o Hudson, o qual possui integração com diversas ferramentas de teste e *feedback*, e contém uma imensa gama de *plugins* disponíveis.
- **Scripts de Construção:** Também chamado de Gerenciador de Construção, constitui de um simples *script*, ou conjunto do mesmo, que é usado para compilar, testar, inspecionar e disponibilizar o software de maneira automatizada. Não foi utilizada uma única ferramenta para esta fase do processo (como o Ant), mas sim diversos *scripts* escritos em Python, que supriram as necessidades do projeto.
- **Mecanismos de feedback:** Não existe um mecanismo padrão para oferecer tal recurso. Pode-se utilizar e-mails, SMS, Twitter, ou mesmo dispositivos físicos como sirenes. Devido a integração do Hudson com o servidor de e-mail, mensagens sobre os *status* das construções, relatórios de inspeção e resultado dos testes eram enviadas ao final de cada integração.
- **Ferramentas de Teste:** Não necessariamente uma ferramenta de IC, mas de extrema importância para qualquer sistema. Para o projeto realizado na Fundação, foram automatizados os testes unitários, com o uso de DocTests e PyUnit, e os testes funcionais, com o uso de um componente do *framework web* utilizado.
- **Ferramentas de Análise de Código:** Automatizar inspeção de código, para verificar, por exemplo, aderência as normas pré-estabelecidas, a complexidade ciclomática, modularidade, duplicação de código entre outras. Como a linguagem de programação utilizada no projeto foi Python, o Pylint se mostrou um recurso interessante para a análise estática de código, verificando a aderência aos padrões especificados pela PEP0008 [Rossum 2001].

A tabela 1 apresenta as ferramentas utilizadas no ambiente de desenvolvimento.

TABELA 1. FERRAMENTAL UTILIZADO NO AMBIENTE DE DESENVOLVIMENTO

Sistema de controle de versão	Subversion (SVN)	subversion.apache.org
Servidor de IC	Hudson	hudson-ci.org
Scripts de Construção	Scripts	Python python.org
Mecanismos de *feedback*	E-mail	www.postfix.org
Ferramentas de Teste	DocTests e PyUnit	pyunit.sourceforge.net
Ferramentas de Análise de Código	Pylint	www.logilab.org/857

4.2. Dia-a-dia da IC

No projeto realizado na Fundação, a adoção do processo de IC mudou a maneira como alguns desenvolvedores se comportavam em relação ao código, pois os mesmos sa-

biam que caso não realizassem os testes automatizados ou não aderissem aos padrões de codificação, os relatórios gerados após cada integração mostrariam o problema. Isso gerou um certo desconforto por parte dos desenvolvedores no início do projeto, pois cada um possuía um estilo próprio de codificar. No entanto, o número reduzido de pessoas e a prática de programação em par fez com que, rapidamente, a equipe se adequasse aos padrões exigidos (PEP0008).

Assim, após inserir um novo trecho de código ou alterar algum existente, os desenvolvedores executavam os *scripts* de construção (compilação, teste e inspeção) localmente. Caso não ocorressem problemas durante a construção local, cada desenvolvedor submetia os artefatos inseridos/modificados/excluídos para o SCV. Em períodos pré-determinados, o servidor de IC, utilizando o auxílio da ferramenta *cron*, verificava se havia ocorrido alguma alteração no projeto. Tal período foi determinado calculando-se o tempo médio do intervalo entre *commits* ao SCV, que era em torno de 20 minutos. Caso houvesse modificações, o servidor recuperava todos os artefatos do projeto e iniciava a execução dos *scripts* de construção, dessa vez com todos os componentes integrados. Ao final do processo, o servidor de IC enviava um e-mail com informações sobre o status da construção a todos os desenvolvedores envolvidos no projeto, deixando-os cientes do produto "final" obtido.

5. RESULTADOS

No projeto realizado pela equipe de desenvolvimento da FAPEAL, vários benefícios foram encontrados com a adoção da prática de IC, dos quais podemos citar:

1. Maior confiabilidade na realização de alterações no código, dado a constante inspeção e testes realizados neste artefato;
2. Maior visibilidade do projeto como um todo, visto que os relatórios e a documentação do projeto passaram a ser gerados automaticamente no processo de construção através de ferramentas como o Epydoc, Cobertura e plugins do SVN, como o WebSVN e StatSVN;
3. Ganho de produtividade, dado que algumas tarefas manuais passaram a ser automatizadas, como a inserção de dados de teste na base de dados e o *deploy* do projeto no ambiente de homologação, permitindo que os desenvolvedores se preocupassem com as regras do negócio, de fato;
4. Aumento da qualidade do código, devido a presença de analizadores estáticos;
5. Maior estabilidade nos *releases* gerados, visto que os mesmos só eram entregues caso se apresentassem completamente funcionais.

Apesar dos benefícios citados acima, algumas dificuldades foram encontradas durante as fases de adoção e utilização das práticas de IC, como a definição de um modelo estrutural que se adequasse a realidade do projeto, o processo de configuração inicial das ferramentas utilizadas e a grande quantidade de construções que falharam no início do projeto.

6. CONCLUSÕES

Diante do que foi apresentado com o desenvolvimento deste trabalho, ficaram claras as vantagens obtidas pela adoção da prática IC no desenvolvimento de software, pois esta se apresen-

tou como uma ferramenta que contribuiu para satisfazer às necessidades da Fundação no que se refere à velocidade e qualidade da entrega do produto final do projeto.

Vale a pena ressaltar que a utilização de ferramentas livres na estratégia de IC se fez de suma importância, pois além de proporcionar as vantagens intrínsecas ao software livre, como a redução de custos e o apoio ativo da comunidade, a utilização de ferramentas livres agrega ainda mais aos valores que a IC traz aos projetos, aumentando o grau de confiança da equipe sobre o produto final desenvolvido.

Como trabalho futuro, pretende-se continuar com a utilização da prática de IC no desenvolvimento de outros projetos internos à Fundação, a fim de se obter uma análise mais detalhada dos resultados de sua adoção a partir do exame de um maior número de artefatos e métricas de software para extração de dados mais formais.

REFERÊNCIAS

Cockburn, A. (2009). *Crystal Clear: A Human-Powered Methodology for Small Teams*. Addison-Wesley, 1 edition.

Collins-Sussman, B., Pilato, C. M., and Fitzpatrick, B. W. (2008). *Version control with subversion*, volume 4. O'Reilly Media, Inc.

Duval, P. M., Matyas, S., and Glover, A. (2007). *Continuous Integration*, volume 26. Addison-Wesley.

Fowler, M. (2006). *Continuous integration*.

Kawalerowics, M. and Berntson, C. (2010). *Continuous Integration in* .NET. Manning.

Kim, S., Park, S., Yun, J., and Lee, Y. (2008). Automated Continuous Integration of Component-Based Software: An Industrial Experience. *2008 23rd IEEE/ACM International Conference on Automated Software Engineering*, pages 423–426.

Lindgren, R., Henfridsson, O., and Schultze, U. (2004). Design principles for competence management systems: A synthesis of an action research study. *MIS QUARTERLY*, 28(3):435–472.

Rossum, G. V. (2001). Style guide for python code. http://www.python.org/dev/peps/pep-0008/. Acessado em: Dezembro de 2009.

Saff, D. and Ernst, M. (2004). An experimental evaluation of continuous testing during development. In *Proceedings of the 2004 ACM SIGSOFT international symposium on Software testing and analysis*, page 85. ACM.

Schwaber, K. (2001). *Agile Software Development with Scrum*. Series in Agile Software Development. Prentice Hall, 2 edition.

Sommerville, I. (2007). *Engenharia de Software*. Tradução de Selma Shin Shimizu Melnikoff. Addison Wesley, São Paulo, 8 edition.

Stringer, E. T. (1996). *Action Research: A Handbook for Practitioners*. Sage.

Teles, V. M. (2004). *Extreme Programming: Aprenda como encantar seus usuários desenvolvendo software com agilidade e alta qualidade*. Novatec.