Universidade Federal do Ceará

Campus Quixadá

Curso de Engenharia de Software

Guylherme Tabosa Cabral

Avaliação do impacto de uma Ferramenta de Integração Contínua na Manutenibilidade do software de uma empresa

Quixadá, Ceará

2014

Sumário

Introdução

O presente trabalho visa aplicar no Núcleo de Práticas em Informática da Universidade Federal do Ceará do campus de Quixadá (NPI) a utilização de uma ferramenta de Integração Contínua (IC), buscando melhorar as taxas de manutenibilidade dos sistemas de software produzidos e avaliar a aceitação e facilidade por parte dos desenvolvedores acerca da ferramenta supracitada.

O NPI é o local onde estudantes que estão em ano de conclusão de curso podem estagiar e aprimorar seus conhecimentos adquiridos no decorrer do curso além de concluir seus componentes curriculares obrigatórios. O NPI surgiu devido à pouca demanda de empresas de Tecnologia da Informação (TI) na região onde a universidade se encontra, em Quixadá no Ceará. Dentro do NPI, os projetos desenvolvidos têm como objetivo construir soluções que facilitem as atividades do cotidiano da universidade, esta que tem um grande interesse no desenvolvimento destes projetos, pois consegue reduzir custos ao priorizar construções de sistemas internamente. E em paralelo, aumentar a qualidade dos profissionais formados por ela, além de proporcionar um ambiente real de trabalho que facilita a entrada dos concludentes no mercado de trabalho (GONÇALVES et al., 2013).

Segundo Sommerville (2011, p. 170) "a manutenção de software é o processo geral de mudança em um sistema depois que ele é liberado para uso". Sendo assim entende-se como manutenção de software qualquer alteração realizada no sistema após este ser considerado "pronto"e implantado em seu ambiente de operação. Atualmente, a manutenção de software vem ganhando uma maior atenção por parte das empresas desenvolvedoras de software. Isso acontece devido à maior parte dos gastos de um produto de software estar na fase de manutenção(PIGOSKI, 1997 apud FIGUEIREDO et al., 2005)(POLO; PIATTINI; RUIZ, 2002 apud FIGUEIREDO et al., 2005). Custos esses que segundo Pressman (2010) equivalem a cerca de 70% do esforço total aplicado no projeto.

Mudanças no software são inevitáveis, e não possuem regra, simplesmente acontecem. Garantir que essas mudanças sejam devidamente controladas, identificadas é o papel da Gerência de Configuração (GC). A importância da GC fica evidenciada quando diferentes modelos de maturidade o abordam como o MPS.BR no nível F e o CMMI(Capability Maturity Model Integration) (FURLANETO, 2006).

A experiência em aplicar ferramentas de gestão de configuração com o objetivo de melhorar a manutenibilidade de uma fábrica de software é abordada através de um conjunto de ferramentas automatizadas que buscam melhorar a manutenibilidade do software (OLIVEIRA; NELSON, 2005). Diferentemente do trabalho proposto que busca verificar o impacto de uma ferramentas em específico de Integração Contínua para a

melhoria da manutenibilidade.

Entende-se Integração Contínua como uma ferramenta de auxílio aos desenvolvedores que permite que as mudanças realizadas no software sejam imediatamente avaliadas,testadas,verificadas de modo à prover um *feedback* imediato para correção de possíveis erros de integração que somente seriam verificados futuramente após problemas mais complexos de integração(DUVAL; MATYAS; GLOVER, 2007).

Furlaneto (2006) relata a construção de uma ferramenta própria e um modelo de processo, que apoiasse a gerência configuração, diversas ferramentas foram avaliadas e comparadas com a ferramenta desenvolvida, verificando assim, que a ferramenta desenvolvida atendeu bem o escopo definido pelo MPS.BR (Melhoria de Processo Brasileiro) e após ser comparada com as ferramentas existentes, esta conseguiu atender bem os requisitos definidos. O trabalho citado difere do proposto ao criar uma ferramenta que apoie a Gerência de Configuração, enquanto o proposto visa avaliar o impacto de uma ferramenta específica em projetos e pessoas.

A justificativa pela escolha do tema se dá pela ausência de pesquisas que tenham como objetivo avaliar o impacto que uma ferramenta de integração contínua exerce sobre a manutenibilidade de um produto de software. Gerar conhecimento para o mercado de modo que ajude empresas a avaliar a necessidade e a viabilidade do uso de ferramentas deste tipo nas empresas.

1 Trabalhos Relacionados

Oliveira e Nelson (2005) relata em seu trabalho, o uso de ferramentas de gerência de configuração e *frameworks* de teste unitário em uma fábrica de software. Para isso foram utilizadas cinco ferramentas automatizadas. Primeiramente foi analisado o cenário atual da fábrica de software, e proposto um novo modelo. Dois projetos da empresas desenvolvidos em C++ e ambiente linux foram utilizados neste trabalho. Onde um projeto que estava sendo mantido possuía aproximadamente 140 mil linhas de código e o outro projeto possuía aproximadamente 675 mil linha de código. Após a integração das ferramentas, no cenário antigo os desenvolvedores somente testavam os casos de teste que eles achavam haviam sido afetado pelas mudanças realizadas, no cenário proposto todo os casos de teste eram reexecutados, o resultados obtidos mostraram que para o primeiro projeto apenas 40% dos casos de testes tiveram sucesso, e no outro projeto 8%, após o uso da ferramenta os número aumentaram para 70 % para o primeiro projeto e 95% para o segundo. Além disso foi comparado o tempo que a build fica quebrada antes e depois do uso das ferramentas, verificou-se que antes da introdução da ferramenta a build ficava uma média de 2,55 dias quebrada e depois da implantação da ferramenta esta passou a ficar em média 1 dia quebrada, já o outro projeto não foi possível verificar os dados pois este não havia build. Outro ponto observado foram as quantidades e tamanho das integrações antes da introdução da ferramentas.As integrações eram feitas apenas uma vez ao dia, e com muitas alterações, antes da implantação da ferramenta a média de linhas alteradas estavam em cerca de 170 linhas por alteração enquanto depois do uso da ferramenta passou a ser 66 em média.

2 Objetivos

2.1 Objetivo Geral

Implantar e avaliar os impactos que o uso de uma ferramenta de integração contínua exerce sobre a manutenibilidade de um software.

2.2 Objetivos Específicos

- Avaliar o nível manutenibilidade do software produzido.
- Implantar um processo de utilização de ferramenta de integração contínua.
- Avaliar resultados obtidos.

3 Fundamentação Teórica

3.1 Manutenção de Software

A manutenção de software é qualquer alteração em um sistema de software após a implantação em seu ambiente de operação. Todo software passa por mudanças seja para adaptar-se a outro sistema operacional, mudança de requisitos, ou simplesmente correção de funcionalidades, o sistema irá mudar. Atualmente as empresas estão tendo uma maior atenção ao processo de manutenção de software segundo (PIGOSKI; POLO; PIATTINI; RUIZ, 1997, 2002 apud FIGUEIREDO et al., 2005). Este custo não se restringe a apenas termos financeiros, bem como retrabalho. O esforço de retrabalho ocorre pelo fato da maioria das equipes de manutenção não estarem relacionadas com a equipe de desenvolvimento e pelo fato da pouca atenção com a documentação do software, com o decorrer do tempo evoluções no software são realizadas e a documentação não é devidamente atualizada.

3.1.1 Tipos de Manutenção

Como citado anteriormente todo software passa por mudanças, e essas mudanças sendo realizadas após a entrega do software caracteriza a atividade de manutenção. Um software não se desgasta como peças de um equipamento,mas se deteriora no sentido de os objetivos de suas funcionalidades cada vez menos se adequarem ao ambiente externo. Paduelli (2007, p. 33)

Os tipos de manutenção definidas são: *corretivas*, *adaptativa* e *perfectivas* (LIENTZ; SWANSON, 1980 apud PADUELLI, 2007)

3.1.1.1 Manutenção Corretiva

Manutenções corretivas visam corrigir defeitos funcionais, onde uma determinada funcionalidade do sistema se comporta de maneira diferentes da especificada para aquela funcionalidade. (PFLEEGER, 2001 apud PADUELLI, 2007) relata um problema onde havia um problema de impressão de um relatório, onde as linhas que era impressa por cada folha era maior do que foi especificado sobrepondo as informações das outras linhas, o problema foi identificado como uma falha de driver da impressora, com isso necessitou que o menu de impressão fosse alterado de modo que um novo parâmetro fosse adicionado, este que iria referenciar o número de linhas que seria impressa.

3.1.1.2 Manutenção Adaptativa

As manutenções do tipo adaptativa retratam a alteração do software de modo à adaptar o software a um novo ambiente de execução. Por exemplo alterar o sistema por conta de uma nova lei, que força que o sistema de **adapte** a ambiente externo(PFLEEGER, 2001 apud PADUELLI, 2007) aborda uma situação onde havia um Sistema de Gerência de Banco de Dados (SGBD) foi atualizado, e as rotinas de acesso ao disco também foram alterada necessitando de uma parâmetro a mais. Este tipo de manutenção tem como característica não a correção de defeito, até por que não existia, mas adaptá-lo ao novo ambiente de operação.

3.1.1.3 Manutenção Perfectiva

Manutenções perfectivas tem como objetivo adicionar funcionalidades a mais ao sistemas, seja por obter um *business value* maior ao produto de software, seja para competir com um software concorrente no mercado, ou simplesmente por uma solicitação do usuário, mediante uma avaliação prévia do sistema de modo a avaliar se a arquitetura do sistema suporta as novas funcionalidades sem degradar.s

3.1.2 Custos de Manutenção

Para toda empresa, quanto menos custos melhor, mas o cenário atual nos diz que os maiores custos em produto de software estão na fase de manutenção(PIGOSKI; POLO; PIATTINI; RUIZ, 1997, 2002 apud FIGUEIREDO et al., 2005) e não alterar o software de maneira rápida o suficiente pode gerar grandes prejuízos. Os altos custo relacionados a manutenção estão inerentes a manutenção em si, esta atividade lida com diversos problemas com má documentação do software a ser mantido, e a imprevisibilidade, não saber em que estado o sistema foi construído, sobre que técnicas, padrões, metodologias, etc.

3.1.3 Documentação de Manutenção

junto de ferramentas que auxiliam no processo de evolução. Portanto "o propósito do processo de Gerência de Configuração é estabelecer e manter a integridade de todos os produtos de trabalho de um processo ou projeto e disponibilizá-la a todos os envolvidos."Softex (2013, p. 18)

3.1.4 Plano de Gerenciamento de Configuração

O Plano de Gerenciamento de Configuração (PGC) descreve todas as atividades de configuração e mudança que serão realizadas durante o projeto. Um conjunto de atividades, responsabilidades, ferramentas, recursos e etc. A gerência de configuração tem como

objetivo garantir a integridade dos itens de configuração [1] , através do versionamento, da identificação, controlando mudanças e acesso.

3.2 Gerência de Configuração

A gerência de configuração é a área da engenharia de software responsável pela evolução do software, a gerência de configuração atua durante todo o ciclo de vida do produto de software, e através de técnicas, ferramentas e metodologias, visa garantir que as mudanças que irão ocorrer dentro do ciclo de vida sejam identificadas, avaliadas e comunicada a todos os envolvidos.

3.2.1 Sistema de Controle de Versão

O sistema de controle de versão: "[...] combina procedimentos e ferramentas para gerenciar diferentes versões de objetos de configurações que são criadas durante o processo de engenharia de software"Pressman (2010, p. 927). Atualmente, o uso de sistema de controle de versão se tornou comuns nas empresas de grande e pequeno porte, estas ferramentas permitem que se tenha o controle de diferentes versões de arquivos que estão submetidos ao versionamento, recuperar versões antigas, visualizar alterações realizadas em arquivos e saber por quem e quando o arquivo foi alterado. Através de comandos (i.e.,*check-in*,*check-out*) os usuários conseguem se comunicar com o repositório a fim de obter os artefatos ali armazenados. (MENEZES, 2011) Em situações especais faz-se necessário que os desenvolvedores trabalhem em una linha diferente da original chamada de *mainline*,geralmente essa situação ocorre quando tem-se como objetivo consertar bugs de versões anteriores do repositório, nesse caso um *branch*[2]. A figura acima demonstra a

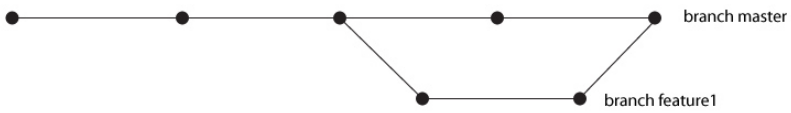

Figura 1 – Branch no Sistema de Controle de Versão

Fonte:(EIS, 2012)

criação de um *branch* paralelo a linha de desenvolvimento principal chamada de branch feature1 e branch master respectivamente, posteriormente ocorre a integração das ações realizar no *branch feature1* é incorporado ao *branch master*.

[1] Itens de configuração são qualquer artefato que esteja sob custódia da Gerência de Configuração
[2] Uma ramificação na linha de desenvolvimento do controle de versão que permite o trabalho em paralelo sobre o mesmo repositório.

3.2.1.1 Sistema de Controle de Versão Local

Um sistema de controle de versão local armazenam todas as informações de um arquivo submetido ao versionamento na máquina local, guardando diferentes versões daquele arquivo localmente como demonstrado na figura abaixo:

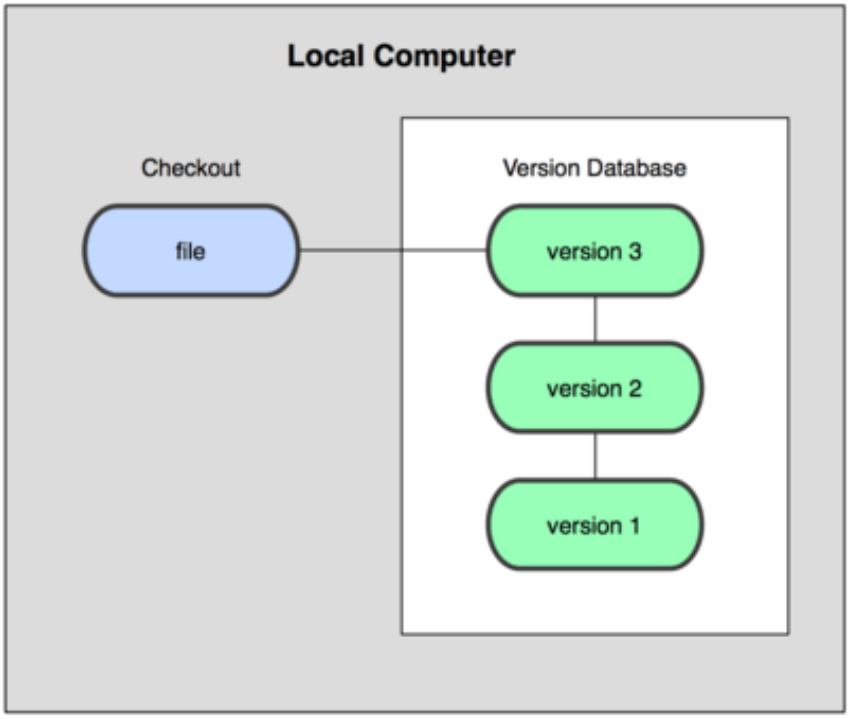

Figura 2 – Sistema de Controle de Versão Local

Fonte:(GIT, S/N)

3.2.1.2 Sistema de Controle de Versão Centralizado

Sistema de controle de versão centralizado como o nome diz possuem um único servidor centralizado, como o *subversion* [3], *perforce* [4], este tipo de padrão de SCV mantém em seu único servidor todos os arquivos versionados. Para cada comando de comunicação com os arquivos versionados uma requisição a esse servidor deverá ser feita, podendo gerar lentidão ou deixar o servidor fora de funcionamento. No exemplo acima dois desenvolvedores trabalhando em máquinas diferentes realizam a comunicação com o servidor central para obter o artefato de trabalho.

3.2.1.3 Sistema de Controle de Versão Distribuído

Os sistemas de controle de versão distribuído possuem um servidor central onde os arquivos são submetidos a versionamento, mas cada desenvolvedor possui em sua máquina

[3] http://subversion.apache.org
[4] http://www.perforce.com

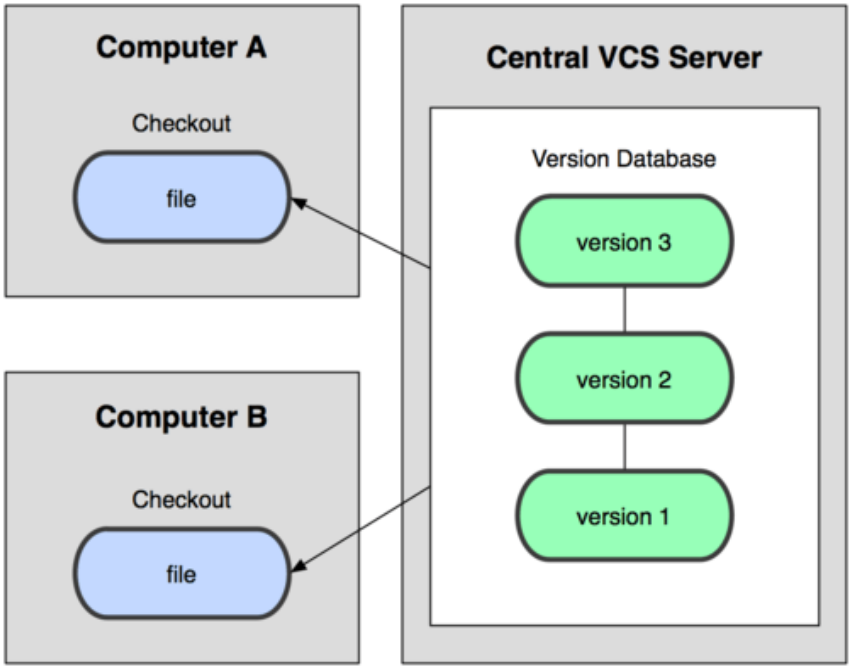

Figura 3 – Sistema de Controle de Versão Centralizado

Fonte:(GIT, S/N)

de trabalho as versões que estavam no servidor, tornando cada *workstation* um "servidor", portanto caso o servidor central orro um problema onde os arquivos sejam perdidos, estes podem ser recuperados via *workstation*, mantendo a integridade dos arquivos e evitando ser um ponto único de falha, como mostra a figura abaixo:

3.2.2 Sistema de Controle de Mudança

Todo software sofre mudanças, lidar com as mudanças é o papel da gerência de configuração, e para isso o gerente de configuração utiliza de sistemas de controle de mudança, o controle de mudança combina procedimentos humanos e ferramentas automatizadas para proporcionar um mecanismo de controle de mudança". Pressman (2010, p .930) As mudanças devem ser avaliadas com cautela baseando-se no custo benefício da mudança, uma combinação de esforço e *business value*. A mudança tem inicio quando um "cliente"solicita a mudanças através de um formulário, conhecida com *change request* onde nesse formulário é descrito os aspectos da mudança, após a solicitação esta deve ser avaliada, verificado se esta mesma mudança já foi solicitada, ja foi corrigida em caso de *bugs*. Após a mudança ser validada, uma equipe de desenvolvedores e avaliaram os impactos que esta mudança têm sobre o sistema, verificação de custo/benefício e esforço de realização. (SOMMERVILLE, 2011) Após essa análise, a mudança será avaliada por um comitê de controle de mudança (CCB) que avaliará o impacto da mudança da perspectiva do negócio, que decidirá se esta mudança será revisada, aprovada ou reprovada. Alguns

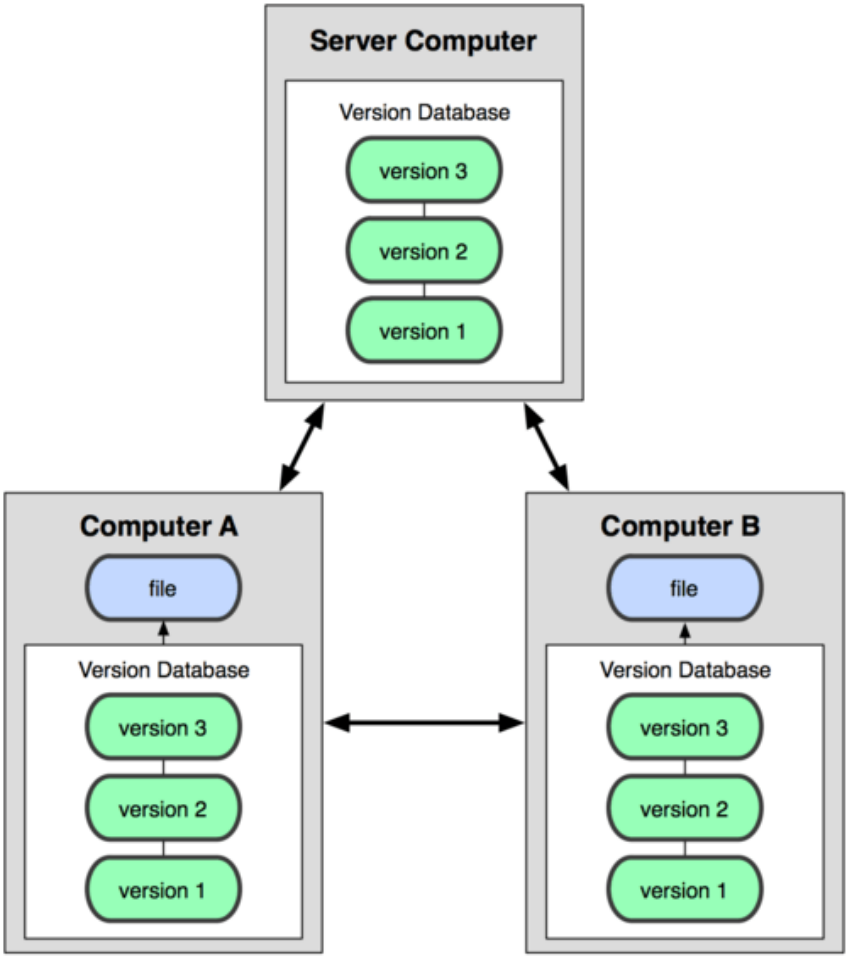

Figura 4 – Sistema de Controle de Versão Distribuído

Fonte:(GIT, S/N)

sistemas que fornecem este controle sobre as mudança são: *redmine* [5], *GitHub* [6] *Jira* [7]

3.2.3 Auditoria de Configuração

Uma auditoria de configuração de software complementa a revisão técnica formal ao avaliar um objeto de configuração quanto às características que geralmente não são consideradas durante a revisão.Pressman (2010, p .934) s Ela tem como objetivo garantir que mesmo com as mudanças realizadas a qualidade foi mantida. As auditorias se dividem em dois tipos: auditorias funcionais e auditorias físicas, a auditoria funcional baseia-se em verificar se os itens de configuração estão devidamente atualizados e se as práticas e padrões foram realizados da maneira correta, enquanto a auditoria funcional, busca verificar os aspectos lógicos dos itens de configuração.

[5] http://www.redmine.org
[6] http://www.github.com
[7] https://www.atlassian.com/software/jira

3.2.4 Ferramentas de Build

As ferramentas de build tem como objetivo automatizar processos repetitivos, aumentando a produtividade e facilitando o trabalho do desenvolvedor. Através da definição de uma rotina, ou conjunto de comandos, o desenvolvedor informa a ferramenta que tipo de processo ele deseja automatizar, pode ser desde compilar e testar uma classe, como dropar e criar uma tabela nova no banco de dados, comprimir arquivos css e javascript, cabe ao desenvolvedor definir o escopo da automatização. Alguns exemplo deste tipo de ferramenta são: *Ant, Grunt, Gulp, Maven.*

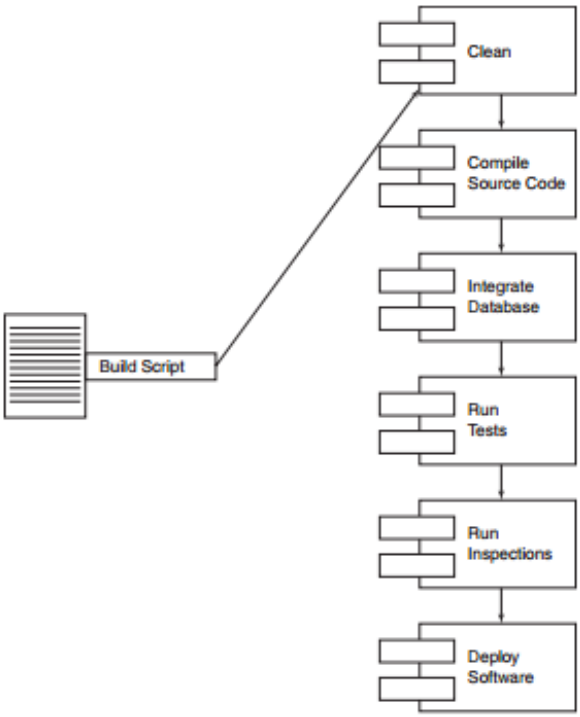

Figura 5 – Processo Lógico de uma Build

Fonte:(DUVAL; MATYAS; GLOVER, 2007)

Na figura acima um script foi definido para realizar as seguintes funções, irá ser feito um clean no projeto, compilará o código fonte, integrará com o banco de dados, executará testes e inspeções no código e por fim irá dar o *deploy* da aplicação.

3.3 Integração Contínua

A integração contínua tem como objetivo identificar erros o mais rápido possível, ela permite que alterações efetuadas e integradas aos repositórios dos sistemas de controle de versão (SCV) sejam posteriormente verificadas e caso erros ocorram, este serão notificados imediatamente ao autor da alteração. Entende-se Integração Contínua como:

> "[...] uma prática de desenvolvimento de software onde os membros de um time integram seu trabalho frequentemente, geralmente cada pessoa integra pelo menos diariamente – podendo haver múltiplas integrações por dia. Cada integração é verificada por uma build automatizada (incluindo testes) para detectar erros de integração o mais rápido possível. Muitos times acham que essa abordagem leva a uma significante redução nos problemas de integração e permite que um time desenvolva software coeso mais rapidamente."Fowler (2000, p. s/n, tradução nossa)

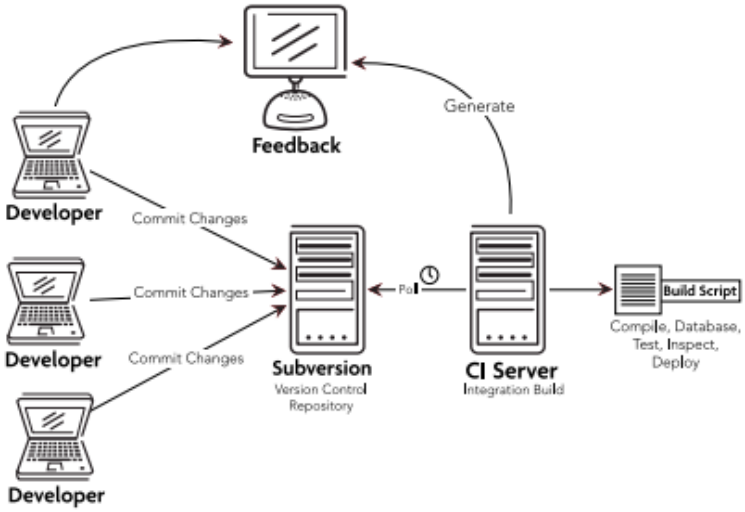

Figura 6 – Ambiente de Integração Contínua

Fonte:(DUVAL, s.n)

A figura acima descreve um ambiente em que um servidor de integração contínua é utilizado. Existem três ambientes de trabalho distintos formado por três desenvolvedores que obtiveram uma cópia do projeto do repositório do SCV para trabalharem em suas *workstation*, durante o trabalho alterações foram realizadas e commitadas ao repositório central, após a inserção junto ao repositório o servidor de integração contínua verifica as alterações e executa uma build de integração,caso exista um problema com a build e esta quebre, o responsável pela alteração será informado sobre a quebra e terá como objetivo consertar a build.

As principais vantagens em utilizar um servidor de integração contínua segundo Duval, Matyas e Glover (2007, p. 29) são:

- Redução de Riscos.
- Redução de processos manuais repetitivos.
- Permitir melhor visibilidade do projeto.
- Estabelecer uma maior confiança no produto do time de desenvolvimento.

3.3.1 Builds Automatizadas

Builds são rotinas de execução definidas com o objetivo de reduzir processos repetitivos. Durante o processo de desenvolvimento de um software muitas ações tendem a serem repetidas por parte dos desenvolvedores, utilizar o tempo para a realização de atividades que poderiam ser automatizadas, de forma manual, reduz a produtividade e preocupações com melhorias devido ao tempo "apertado". Somando-se a isso, uma build garante que tudo que está nela definido será executado, evitando assim, que determinada ação seja esquecida, ou caso um novo membro entre na equipe uma explicação do que ele deve fazer, ou não esquecer de fazer, não faz-se necessário.

3.3.2 Integração Contínua Manual

Na IC manual o processo de integração é realizado individualmente, possibilitando que apenas um desenvolvedor realize check-in no repositório durante o intervalo de integração Menezes (2011). Este tipo de abordagem como permite que apenas uma pessoa realize o *check-in*, as integrações serão contínuas e seguidas e não paralelas, este tipo de abordagem garante uma maior confiabilidade das integrações, pois segue um padrão de integração, os itens do repositório possuem maior consistência e a garantia da estrutura do repositório é mantida. (MENEZES, 2011)

3.3.3 Integração Contínua Automatizada

A integração contínua automatizada é auxiliada pelo uso de um servidor de integração contínua, que obtém do controle de versão as alterações realizadas e executada sua build privada afim de verificar possíveis erros gerados por essas modificações. Ver seção 3.3 Figura 6

> "IC Automática possui a vantagem de ser escalável e, deste modo, oferecer maior suporte ao trabalho colaborativo. Com a utilização de Servidores de IC, a responsabilidade de realizar construções da integração é retirada dos desenvolvedores. Portanto, os desenvolvedores podem realizar check-in sem a necessidade de conquistar a vez de integrar. Esse fator é fundamental para que os check-ins continuem sendo verificados sem a necessidade de um desenvolvedor realizar a construção e identificar problemas, resultando na eliminação do gargalo humano Menezes (2011)."

4 Procedimentos Metodológicos

- Identificar e analisar o desenvolvimento das atividades no NPI.
- Identificar as melhores técnicas de estimativas de manutenção e definir e implantar o modelo ao NPI.
- Definir a utilização de uma ferramenta de integração contínua e um modelo de processo que adeque esta.
- Avaliar e definir a ferramenta à ser utilizada.
- Implementar a utilização da ferramenta e coletar resultados.

4.1 Cronograma de Execução

ATIVIDADES	2014												
	Mai		Jun		Jul		Ago		Set		Out		Nov
Estudo de Campo	x	x	x										-
Defesa do projeto				x									-
Avaliação do processo do NPI					x	x							-
Definição da ferramenta de integração contínua						x	x						-
Implantação do processo de execução de ferramenta de integração contínua							x	x					-
Estimativa de tamanho do projeto								x					-
(Análise dos Dados)									x	x			-
(Avaliação da Execução)										x	x		-
Revisão final da monografia											x	x	-
Defesa do Trabalho Final												x	-

Figura 7 – Cronograma de Execução

Referências

DUVAL, P. M. Continuous integration: Patterns and anti-patterns. *DZone Refcardz*, n. 1, p. 1–6, s.n. Citado na página 16.

DUVAL, P. M.; MATYAS, S.; GLOVER, A. *Continuos Integration: Improving Software Quality and Reducing Risk*. 1. ed. [S.l.]: Pearson Education, 2007. ISBN 978-0-321-33638-5. Citado 3 vezes nas páginas 4, 15 e 16.

EIS, D. Janeiro 2012. Disponível em: <http://tableless.com.br/introducao-das-premissas-dos-controles-de-versao/>. Citado na página 11.

FIGUEIREDO, S. et al. Apoio a manutenção de software atraves de design rationale em ambientes de manutenção de software TABA. *Workshop de Manutenção de Software Moderna*, n. 1, p. 98–113, 2005. Citado 3 vezes nas páginas 3, 9 e 10.

FOWLER, M. Maio 2000. Disponível em: <http://martinfowler.com/articles/continuousIntegration.html>. Citado na página 16.

FURLANETO, R. *FERRAMENTA DE APOIO A GERÊNCIA DE CONFIGURAÇÃO DE SOFTWARE*. Dissertação (Mestrado) — Universidade Regional de Blumenau, Blumenau, Dezembro 2006. Citado 2 vezes nas páginas 3 e 4.

GIT. S/N. Disponível em: <http://git-scm.com/book/pt-br/Primeiros-passos-Sobre-Controle-de-Vers~ao>. Citado 3 vezes nas páginas 12, 13 e 14.

GONÇALVES, E. J. T. et al. Núcleo de práticas em informática: Contribuindo para a formação em sistemas de informação através do desenvolvimento de projetos de software. *XXXIII CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO (WEI*, Maceió, n. 1, p. 601–610, 2013. Citado na página 3.

LIENTZ, B. P.; SWANSON, E. B. *Software Maintenance Management*. Reading, MA, USA: Addison-Wesley, 1980. Citado na página 9.

MENEZES, G. G. L. de. *OURIÇO:UMA ABORDAGEM PARA MANUTENÇÃO DA CONSISTÊNCIA EM REPOSITÓRIOS DE GERÊNCIA DE CONFIGURAÇÃO*. Niterøi: [s.n.], 2011. Citado 2 vezes nas páginas 11 e 17.

OLIVEIRA, P. A. de; NELSON, M. A. V. Integração de ferramentas para minimizar erros provenientes de efeitos colaterais inseridos durante a manutenção. *Workshop de Manutenção de Software Moderna*, n. 1, 2005. Citado 2 vezes nas páginas 3 e 5.

PADUELLI, M. M. *Manutenção de Software: problemas típicos e diretrizes para uma disciplina específica*. São Carlos: [s.n.], 2007. Citado 2 vezes nas páginas 9 e 10.

PFLEEGER, S. L. *Software Engineering: Theory and Practice*. New Jersey, USA: Presentice Hall, 2001. Citado 2 vezes nas páginas 9 e 10.

PIGOSKI, T. M. *Pratical Software Maintenance*. 1. ed. [S.l.]: Wiley, 1997. 400 p. ISBN 978-0471170013. Citado 3 vezes nas páginas 3, 9 e 10.

POLO, M.; PIATTINI, M.; RUIZ, F. Using a qualitative research method for building a software maintenance methodology. *Software Pratice and Experience*, p. 1239–1260, 2002. Citado 3 vezes nas páginas 3, 9 e 10.

PRESSMAN, R. S. *Engenharia de Software*. [S.l.]: Pearson Education, 2010. 1056 p. ISBN 978-85-346-0237-2. Citado 4 vezes nas páginas 3, 11, 13 e 14.

SOFTEX. *Guia de Implementacao - Parte 2: Fundamentacao para Implementacao do Nivel F do MR-MPS-SW:2012*. [S.l.], 2013. Disponível em: <http://www.softex.br/wp-content/uploads/2013/07/MPS.BR_Guia_de_Implementacao_Parte_2_2013.pdf>. Acesso em: 17.2.2013. Citado na página 10.

SOMMERVILLE, I. *Engenharia de Software*. 9. ed. [S.l.]: Pearson Education, 2011. 529 p. ISBN 978-85-7936-108-1. Citado 2 vezes nas páginas 3 e 13.