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

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

NPI	Núcleo de Práticas em Informática
CMMI	Capability Maturity Model Integration
MPS.BR	Melhoria de Processo Brasileiro
GC	Gerência de Configuração
SCV	Sistema de Controle de Versão
SCM	Sistema de Controle de Mudança
IC	Integração Contínua
UFC	Universidade Federal do Ceará
TI	Tecnologia da Informação
PGC	Plano de Gerenciamento de Configuração
SGBD	Sistemas de Gerência de Banco de Dados
CCB	Change Control Board
SBC	Sociedade Brasileira de Computação
MI	Maintainability Index

Sumário

1 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 o impacto que esta ferramenta possui sobre a manutenibilidade de um sistema de software.

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. Em paralelo, aumenta 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).

Dentro do NPI existe um modelo de processo definido em que os desenvolvedores devem seguir para o exercício de suas atividades (GONÇALVES et al., 2013). Entretanto, este não é devidamente seguido, ocasionando uma despadronização na maneira como estes desenvolvedores trabalham em seus projetos. Somado-se a isto, o NPI apresenta problemas tais como: "Baixa qualidade da documentação dos sistemas; [...] Falta de uma equipe de manutenção; [...] Rotatividade dos profissionais"o que acaba gerando graves problemas de manutenção (PADUELLI; SANCHES, 2006, p. 4).

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, esta vem ganhando uma maior atenção por parte das empresas desenvolvedoras de software. Isso acontece devido aos altos custos na fase de manutenção, podendo atingir 70% do esforço total aplicado no projeto, além de sofrer possíveis aumentos ao longo da produção do software (PRESSMAN, 2010).

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 a ser desenvolvido nesse projeto, que busca verificar o impacto de uma ferramenta em específico, de Integração Contínua, na melhoria da manutenibilidade.

Entende-se Integração Contínua como uma ferramenta de gestão de configuração, que auxilia os desenvolvedores e permite que as mudanças realizadas no software sejam imediatamente avaliadas, testadas, verificadas de modo a 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).

A justificativa pela escolha do tema se deu através da ausência de pesquisas que busquem avaliar o impacto que o uso de uma ferramenta de integração contínua utilizada durante desenvolvimento de um produto de software exerce sobre a manutenibilidade do software produzido. As buscas por pesquisas foram realizadas no: acervo da Sociedade Brasileira de Computação (SBC), *Google Academys*. Proporcionar conhecimento para o mercado de modo a ajudar empresas a avaliarem a necessidade, viabilidade do uso de ferramentas deste gênero.

2 Trabalhos Relacionados

3 Objetivos

3.1 Objetivo Geral

Implantar e avaliar o impacto que um ferramenta de integração contínua exerce sobre a manutenibilidade de um software construído no NPI.

3.2 Objetivos Específicos

- Avaliar o nível manutenibilidade do software GAL (Gestão de Aquisição de Livros) produzido no NPI.
- Avaliar curva de aprendizagem e aplicar treinamento adequado para utilização da ferramenta.
- Avaliar o nível de manutenibilidade do software após a introdução da utilização da ferramenta.

4 Fundamentação Teórica

4.1 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 para adaptar-se a outro sistema operacional, mudanças de requisitos, ou simplesmente correção de funcionalidades. Atualmente as empresas estão tendo uma maior atenção ao processo de manutenção de software segundo (**????**, **????** 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 (SOUZA et al., 2005).

4.1.1 Tipos de Manutenção

Como citado anteriormente todo software passa por mudanças, e essas mudanças sendo realizadas após a entrega do software caracterizam 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: (LIENTZ; SWANSON, 1980 apud PADUELLI, 2007) *corretivas*, *adaptativa* e *perfectivas*

4.1.1.1 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 de impressão de um relatório, em que as linhas que eram impressas por cada folha eram maiores do que foi especificado, sobrepondo as informações das outras linhas. O problema foi identificado como uma falha de driver da impressora, e foi preciso alterar o menu de impressão para adicionar um novo parâmetro, este que iria referenciar o número de linhas que seria impresso.

4.1.1.2 Manutenção Adaptativa

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

4.1.1.3 Manutenção Perfectiva

Manutenções perfectivas tem como objetivo adicionar funcionalidades ao sistemas, seja para obter um *business value* maior ao produto de software, para competir com um software concorrente no mercado, ou simplesmente para atender uma solicitação de usuário. Este processo de mundaça é realizado mediante uma avaliação prévia do sistema, que visa avaliar se a arquitetura do sistema suporta as novas funcionalidades sem degradá-la.

4.1.2 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(**????**, **????** 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.

4.1.3 Documentação de Manutenção

4.1.4 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, que são qualquer artefato que esteja sob custódia da Gerência de Configuração, através do versionamento, da identificação, controlando mudanças e acesso.

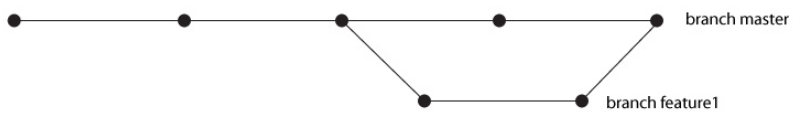
4.2 Gerência de Configuração

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

4.2.1 Sistema de Controle de Versão

Um 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 comum nas empresas de grande e pequeno porte. Tais ferramentas permitem que se tenha o controle de diferentes versões de arquivos que estão submetidos ao versionamento, recuperação de 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*, uma ramificação na linha de desenvolvimento do controle de versão que permite o trabalho em paralelo sobre o mesmo repositório. A figura Figura 1 demonstra a criação de um *branch* paralelo a linha de

Figura 1 – Branch no Sistema de Controle de Versão



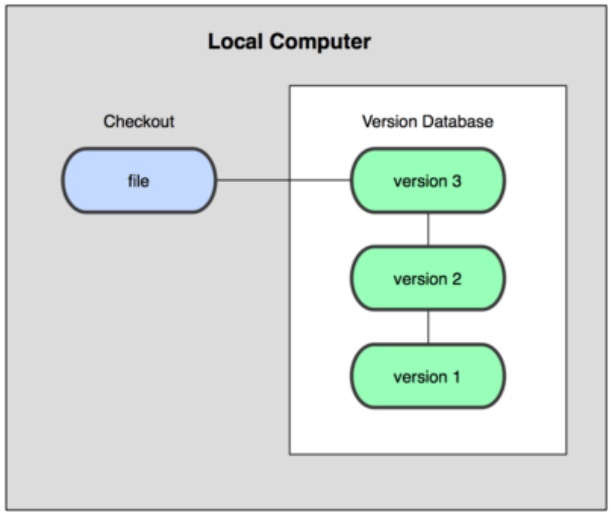
Fonte: Eis (2012)

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*.

4.2.1.1 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 Figura 2.

Figura 2 – Sistema de Controle de Versão Local



Fonte:(GIT, S/N)

4.2.1.2 Sistema de Controle de Versão Centralizado

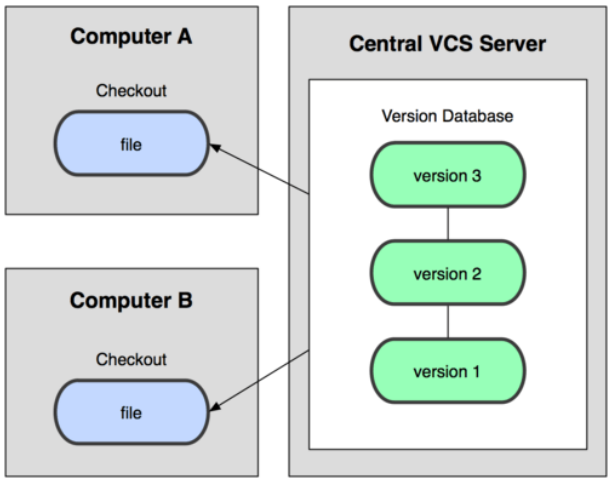
Sistema de controle de versão centralizado como o nome diz possuem um único servidor centralizado, como o *subversion* [1], *perforce* [2], este tipo de padrão de SCV mantém em seu único servidor todos os arquivos versionados. Para cada comando de comunicação realizado nos arquivos versionados, uma requisição 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.

4.2.1.3 Sistema de Controle de Versão Distribuído

Os sistemas de controle de versão distribuído possuem um servidor central onde os arquivos são submetidos a versionamento, entretanto cada desenvolvedor possui em sua máquina de trabalho as versões que estavam no servidor, tornando cada *workstation* um "servidor", portanto, caso ocorra um problema no servidor central, estes podem ser recuperados via *workstation*, mantendo a integridade dos arquivos e evitando ser um ponto único de falha, como mostra a figura Figura 4:

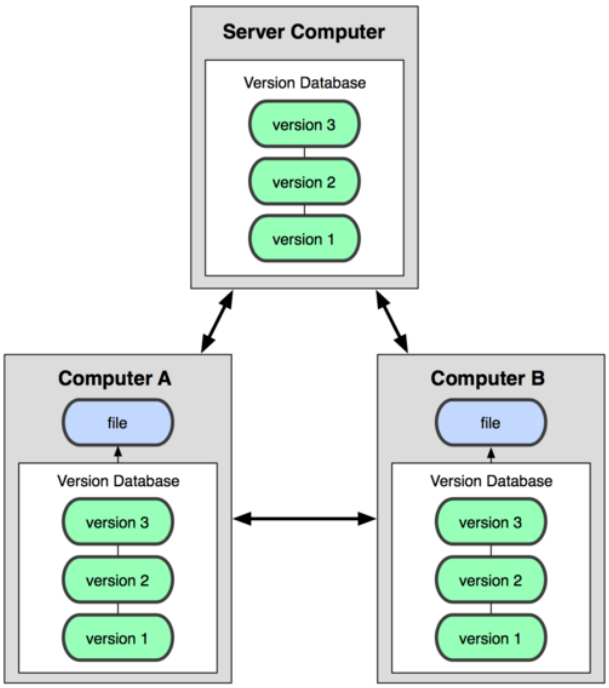
[1] http://subversion.apache.org
[2] http://www.perforce.com

Figura 3 – Sistema de Controle de Versão Centralizado



Fonte: GIT (S/N)

Figura 4 – Sistema de Controle de Versão Distribuído



Fonte: GIT (S/N)

4.2.2 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 um 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, em seu custo benefício, uma combinação de esforço e *business value*. A mudança tem início quando um "cliente"solicita

a mudanças através de um formulário, conhecida com *change request*. Nesse formulário é descrito os aspectos da mudança, após a solicitação ser realizada, esta deve ser avaliada, verificando se a mesma já foi solicitada, ou corrigida em caso de *bugs*. Após a mudança ser validada, uma equipe de desenvolvedores e avaliam os impactos que esta mudança têm sobre o sistema, verificando custo/benefício e esforço de realização (SOMMERVILLE, 2011). Posterior a essa análise, a mudança será avaliada por um comitê de controle de mudança (CCB) que avaliará o impacto da perspectiva do negócio, o que decidirá se esta mudança será revisada, aprovada ou reprovada. Alguns sistemas que fornecem este controle sobre as mudança são: *redmine* [3], *GitHub* [4] *Jira* [5]

4.2.3 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). 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.

4.2.4 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.*

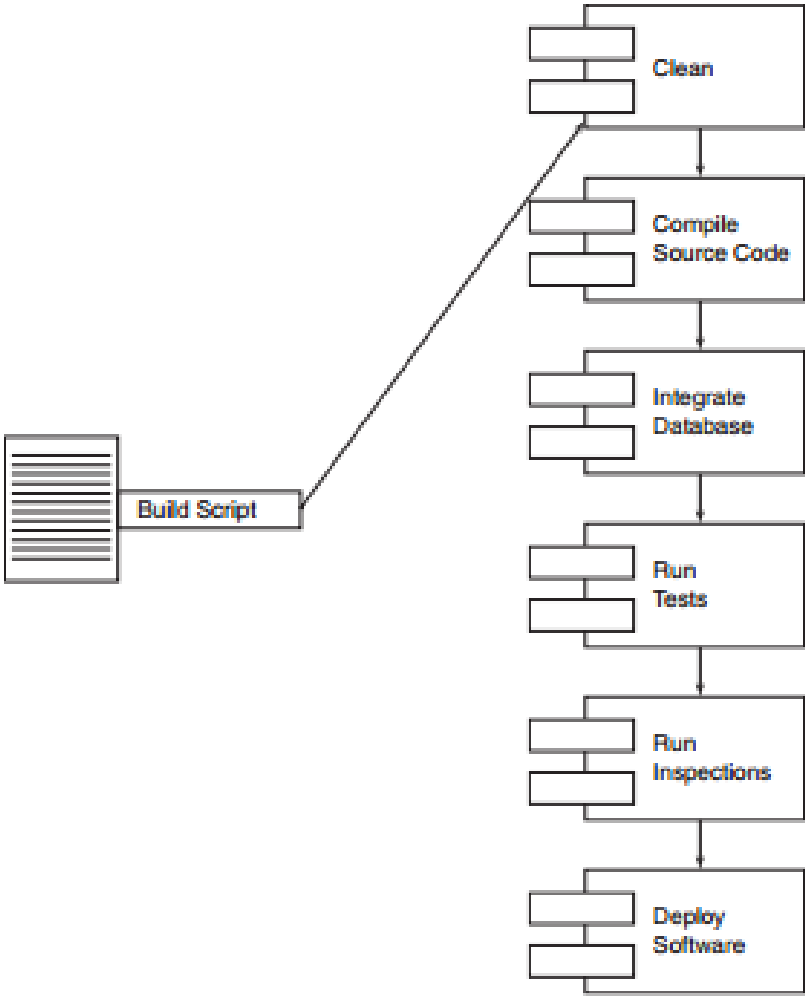
Na figura Figura 5 um script foi definido para realizar as seguintes funções, será realizado um clean no projeto, compilará o código fonte, integrará com o banco de dados, executará testes e inspeções no código e por fim irá dar o *deploy* da aplicação.

4.3 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

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

Figura 5 – Processo Lógico de uma Build



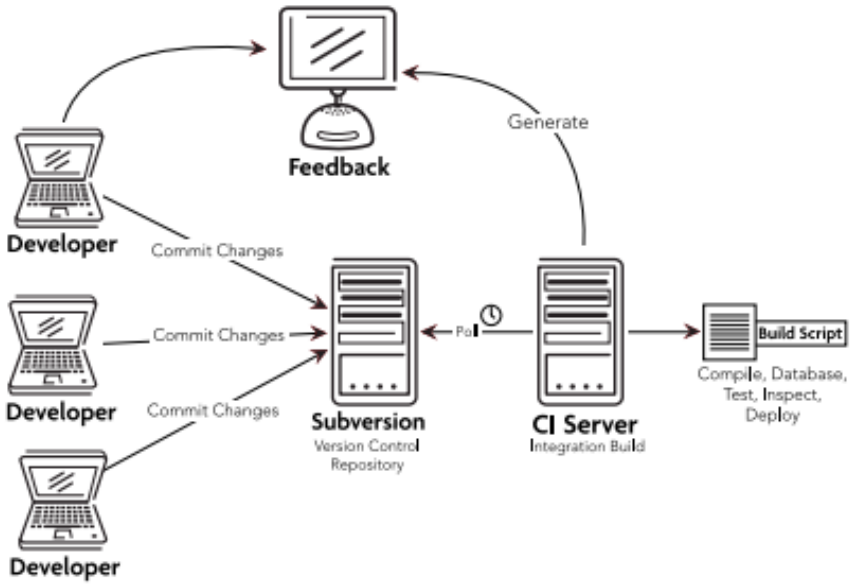
Fonte: Duval, Matyas e Glover (2007)

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, tradução nossa)

A figura Figura 6 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

Figura 6 – Ambiente de Integração Contínua



Fonte: Duval (s.n)

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.

4.3.1 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.

4.3.2 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).

4.3.3 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 4.3 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, p .54).

5 Procedimentos Metodológicos

Esta seção tem como objetivo descrever os passos tomados de modo a concluir os objetivos definidos anteriormente.

- Identificar e analisar o desenvolvimento das atividades no NPI.

A identificação e análise das atividades do NPI serão realizadas por meio de entrevistas com líderes técnicos de equipes, professores supervisores e avaliação do modelo de processo proposto por Gonçalves et al. (2013) com ênfase na atividade de Gerência de Configuração.

- Definir a utilização de uma ferramenta de integração contínua.

A ferramenta utilizada será a CruiseControl [1] por ser uma ferramenta *open source* e por fornecer suporte a diferentes ferramentas de build.

- Avaliar o nível de manutenibilidade do software anteriormente e posteriormente a utilização da ferramenta de integração contínua.

Para avaliação da manutenibilidade so software será utilizado a medida do Índice de Manutenibilidade (Maintainability Index - MI). Para obtenção do MI será preciso a utilização de algumas medidas como o Volume de Halstead e a quantidade de linhas de código. Uma fórmula para o cálculo da manutenibilidade é fornecido pelo trabalho de Vale (2013).

- Implementar a utilização da ferramenta e coletar resultados.

O processo de implementação será por meio de treinamento para explanação da importância da ferramenta seus benefícios e entendimento de sua funcionalidade. Ao final um relatório será gerado para análise dos dados.

5.1 Cronograma de Execução

[1] http://cruisecontrol.sourceforge.net/

Tabela 1 – Cronograma das atividades previstas

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			
Avaliação da Manutenibilidade do GAL				X	X		
Implantação da Integração Contínua				X	X		
Treinamento do uso da Ferramenta					X	X	
Avaliação da Manutenibilidade do GAL						X	
Análise do Dados de Manutenibilidade						X	
Revisão final da monografia						X	X
Defesa do Projeto							X

Fonte: Elaborado pelo Autor

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 6, 15 e 16.

EIS, D. Janeiro 2012. Disponível em: <http://tableless.com.br/introducao-das-premissas-dos-controles-de-versao/>. Acesso em: 27.3.2014. 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 2 vezes nas páginas 9 e 10.

FOWLER, M. Maio 2000. Disponível em: <http://martinfowler.com/articles/continuousIntegration.html>. Acesso em: 27.3.2014. Citado na página 15.

FURLANETO, R. *FERRAMENTA DE APOIO A GERÊNCIA DE CONFIGURAÇÃO DE SOFTWARE*. Dissertação (Mestrado) — Universidade Regional de Blumenau, Blumenau, Dezembro 2006. Citado na página 6.

GIT. S/N. Disponível em: <http://git-scm.com/book/pt-br/Primeiros-passos-Sobre-Controle-de-Vers~ao>. Acesso em: 27.3.2014. Citado 2 vezes nas páginas 12 e 13.

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 2 vezes nas páginas 5 e 18.

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 na página 6.

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.

PADUELLI, M. M.; SANCHES, R. Problemas em manutenção de software caracterização e evolução. *III Workshop de manutenção moderna*, 2006. Citado na página 5.

PFLEEGER, S. L. *Software Engineering: Theory and Practice*. New Jersey, USA: Presentice Hall, 2001. Citado 2 vezes nas páginas 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 5, 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: 17.4.2014. Citado na página 11.

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 5 e 14.

SOUZA, S. C. B. de et al. Documentação essencial para manutenção de software ii. *Workshop de Manutenção de Software Moderna*, n. 1, 2005. Citado na página 9.

VALE, G. A. do. *AVALIAÇÃO DA MANUTENIBILIDADE DE SISTEMAS DE SOFTWARE DERIVADOS DE LINHAS DE PRODUTOS DE SOFTWARE*. 110 p. — Universidade Federal de Lavras, Lavras, 2013. Citado na página 18.