



**Universidade Federal do Ceará**  
**Campus Quixadá**  
**Curso de Engenharia de Software**

**Guyllherme Tabosa Cabral**

**AVALIAÇÃO DO IMPACTO DE FERRAMENTA DE INTEGRAÇÃO CONTÍNUA  
EXERCE NA MANUTENIBILIDADE DO SOFTWARE**

**Quixadá, Ceará**

**2014**

# Lista de ilustrações

Figura 1 – ISO/IEC 9126 . . . . .	13
Figura 2 – Medidas de Halstead. . . . .	14
Figura 3 – Fórmula de Maccabe. . . . .	15
Figura 4 – Índice de Manutenibilidade. . . . .	15
Figura 5 – Branch no Sistema de Controle de Versão . . . . .	17
Figura 6 – Sistema de Controle de Versão Local . . . . .	17
Figura 7 – Sistema de Controle de Versão Centralizado . . . . .	18
Figura 8 – Sistema de Controle de Versão Distribuído . . . . .	19
Figura 9 – Processo Lógico de uma Build . . . . .	21
Figura 10 – Ambiente de Integração Contínua . . . . .	22

# Lista de tabelas

Tabela 1 – Cronograma das atividades previstas . . . . .	26
--	----

# 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

<b>1</b>	<b>INTRODUÇÃO</b>	<b>6</b>
<b>2</b>	<b>TRABALHOS RELACIONADOS</b>	<b>8</b>
2.1	Impacto de Padrões de Projeto na Manutenibilidade	8
2.2	Manutenibilidade em Softwares Advindos de Linhas de Produto de Software	8
<b>3</b>	<b>OBJETIVOS</b>	<b>9</b>
3.1	Objetivo Geral	9
3.2	Objetivos Específicos	9
<b>4</b>	<b>FUNDAMENTAÇÃO TEÓRICA</b>	<b>10</b>
<b>4.1</b>	<b>Manutenção de Software</b>	<b>10</b>
4.1.1	Tipos de Manutenção	10
4.1.1.1	Manutenção Corretiva	10
4.1.1.2	Manutenção Adaptativa	11
4.1.1.3	Manutenção Perfectiva	11
4.1.2	Custos de Manutenção	11
4.1.3	Documentação de Manutenção	11
<b>4.2</b>	<b>Manutenibilidade de Software</b>	<b>12</b>
4.2.1	Métrica de Software	13
4.2.1.1	Métrica de Halstead	13
4.2.1.2	Complexidade Ciclomática de McCabe	15
4.2.1.3	Maintainability Index	15
<b>4.3</b>	<b>Gerência de Configuração</b>	<b>16</b>
4.3.1	Plano de Gerenciamento de Configuração	16
4.3.2	Sistema de Controle de Versão	16
4.3.2.1	Sistema de Controle de Versão Local	17
4.3.2.2	Sistema de Controle de Versão Centralizado	18
4.3.2.3	Sistema de Controle de Versão Distribuído	18
4.3.3	Sistema de Controle de Mudança	19
4.3.4	Auditoria de Configuração	20
4.3.5	Ferramentas de Build	20
<b>4.4</b>	<b>Integração Contínua</b>	<b>20</b>
4.4.1	Builds Automatizadas	22
4.4.2	Integração Contínua Manual	22

4.4.3	Integração Contínua Automatizada . . . . .	23
5	<b>PROCEDIMENTOS METODOLÓGICOS . . . . .</b>	<b>24</b>
5.1	<b>Cronograma de Execução . . . . .</b>	<b>26</b>
	 <b>Referências . . . . .</b>	 <b>27</b>

# 1 Introdução

O presente trabalho visa implantar 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 analisar 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. Este 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 (GONÇALVES et al., 2013). 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.

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), esta que tem sua importância evidenciada quando diferentes modelos de maturidade o abordam como MPS.BR no nível F e o CMMI (Capability

Maturity Model Integration) (FURLANETO, 2006). Para identificar e auxiliar no controle das mudanças, um conjunto de ferramentas CASE (Computer-Aided Software Engineering) são utilizadas pela Gerência de Configuração.

A experiência em aplicar ferramentas de gestão de configuração com o objetivo de melhorar a manutenibilidade de uma fábrica de software é explanado por Oliveira e Nelson (2005). Eles inseriram a utilização destas ferramentas para evitar a inserção de novos erros proveniente de manutenções realizadas no software. 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 Academics* utilizando os termos: *Continuous Integration*, *Continuous Integration and maintainability index*. Proporcionar conhecimento para o mercado de modo a ajudar empresas a avaliarem a necessidade, viabilidade do uso de ferramentas deste gênero.



## 2 Trabalhos Relacionados

Nesta seção será descrito trabalhos que influenciaram os conceitos envolvidos neste trabalho além de demonstrar pontos comuns e distintos entre si e o proposto. A seção 2.1 descreve o trabalho de Ferreira (2012) e a seção 2.2 descreve o trabalho de Vale (2013).

### 2.1 Impacto de Padrões de Projeto na Manutenibilidade

Analisar atributos de qualidade de software faz se importante para quantificar a qualidade de um produto, esta quantificação é obtida por meio de medição. Ferreira (2012) buscou avaliar a taxa de manutenibilidade de um produto de software, através da utilização de métricas específicas para mensurar a manutenibilidade de um software. Com o objetivo de avaliar o impacto da utilização de um conjunto de boas práticas de programação, o autor mensurou o sistema a ser estudado, refatorou utilizando tais práticas, e posteriormente refez as medições.

O trabalho comprovou que a utilização de boas práticas de programação foi eficaz na melhora das taxas de manutenibilidade, que por consequência melhora a qualidade do software.

O trabalho supracitado difere do proposto pois, este refatorou um sistema existente para fins de análise da utilização das boas práticas, entretanto o trabalho a ser desenvolvido não tem como intuito refatorar um sistema legado, mas verificar o impacto que a utilização de um ferramenta de integração contínua exercerá na manutenibilidade de um software ainda em construção, esta sendo inserida durante o desenvolvimento do mesmo, e por fim implantar a utilização deste tipo de ferramenta.

### 2.2 Manutenibilidade em Softwares Advindos de Linhas de Produto de Software

## 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

Nesta seção será apresentado os conceitos necessários para um completo entendimento deste trabalho. A seção 4.1 apresentará uma explanação a cerca de manutenção e posteriormente na seção 4.2 os conceitos que envolvem a manutenibilidade, na seção 4.3 conceitos inerentes a Gerência de configuração, juntamente com algumas ferramentas CASE e com ênfase em Integração Contínua descrito na seção 4.4

### 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 devido aos altos custos que este demanda, cerca de 70% (PRESSMAN, 2010) (PIGOSKI, 1997). 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

Reduzir custo é objetivo de toda empresa, mas o cenário atual nos diz que os maiores custos em produto de software estão na fase de manutenção (PIGOSKI, 1997). 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, como 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

A documentação de software é uma das atividades mais importantes da atividade de manutenção, pois auxilia uma comunicação com uma equipe externa de manutenção (SOUZA et al., 2005). Sua importância torna-se mais evidenciada quando Wong (1995 apud SOUZA et al., 2005) afirma que mais 50% do esforço de evolução do software é aplicado no entendimento do programa e na fase de manutenção, sua estrutura. Souza et

al. (2005) identificaram os artefatos mais utilizadas por mantenedores, segundo eles de vite e quatro artefatos disponíveis, quatorze foram utilizados.

## 4.2 Manutenibilidade de Software

De acordo com a norma ISO/IEC9126-1, existem seis divisões para mensurar a qualidade de um produto. São elas:

- Funcionalidade

É a capacidade do software realizar as funções que ele foi construído para fazer.

- Confiabilidade

"É o nível de confiança que se pode ter no software levando em consideração sua maturidade, sua tolerância a falhas e sua capacidade de se recuperar" (FERREIRA, 2012, p. 28).

- Usabilidade

É o nível de facilidade que o o software possui, baseado em sua inteligibilidade, facilidade de aprendizado e operabilidade (FERREIRA, 2012).

- Eficiência

"Diz respeito ao comportamento do software em relação ao tempo e aos recursos existentes onde quanto maior a otimização dos recursos do sistema em relação a estes atributos, maior é sua eficiência" (FERREIRA, 2012, p. 28).

- Manutenibilidade

É facilidade de o software possui de ser modificado, e com mudanças de baixa propagação de impactos.

- Portabilidade

É a facilidade do software ter seu ambiente de operação alterado.

Dentre estas divisões de qualidade o foco deste trabalho é a manutenibilidade. A manutenibilidade divide-se ainda em mais cinco subcaracterísticas, são elas:

- Analisabilidade

O esforço necessário para identificação de uma deficiência no sistema (FERREIRA, 2012).

- Modificabilidade

É o esforço necessário para realizar modificações no sistema.

- Estabilidade

"[...]Evidencia o risco de acontecerem efeitos inesperados ocasionados por alguma modificação efetuada" (FERREIRA, 2012, p. 29).

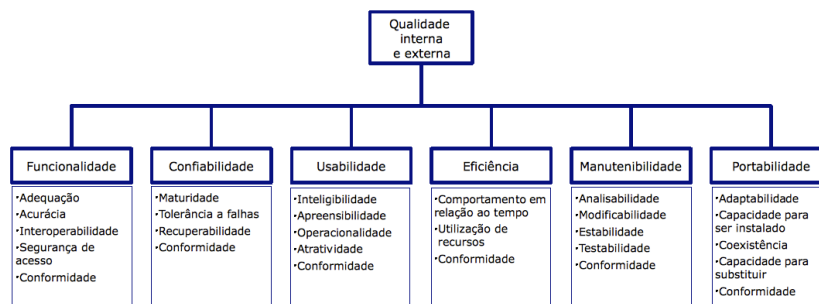
- Testabilidade

É o esforço aplicado para testar e validar alterações realizadas.

- Conformidade

"Analisa se o software está condizente com padrões que permitirão ao mesmo ser alocado em diferentes ambientes. Em outras palavras, se suas “conexões” estão padronizadas" (FERREIRA, 2012, p. 29).

Figura 1 – ISO/IEC 9126.



Fonte: Kichenbrand e Fontes (2012).

## 4.2.1 Métrica de Software

Métricas de software caracterizam-se como uma forma de mensurar atributos de software. Durante o desenvolvimento do produto de software, mensurar atributos de qualidade são importantes pois visam aumentar a qualidade do produto.

### 4.2.1.1 Métrica de Halstead

Halstead (1977 apud FERREIRA, 2012) foi a primeira teoria a associar leis quantitativas ao desenvolvimento de software, por meio de um conjunto de medidas que podem ser

retiraras do código após sua geração. Essas medidas foram definidas como operandos, que são valores que operam alguma instrução, e operadores que são instruções que modificam operandos. Tem-se:

- **n1** - O número de operadores diferentes que aparece em um software;
- **n2** - O número de operandos diferentes que aparece em um software;
- **N1** - O número total de aparição de operadores;
- **N2** - O número total de aparição de operandos;

Através destas medidas tem-se:

Figura 2 – Medidas de Halstead.

Medida	Fórmula
Extensão do programa	$N = N1 + N2$
Vocabulário do programa	$n = n1 + n2$
Volume	$V = N * (\log_2 n)$
Dificuldade	$D = (n1/2) * (N2/2)$
Esforço	$E = D * V$

Fonte: Ferreira (2012).

Onde segundo Ferreira (2012) define-se como:

- Extensão do programa

É a soma de todos os operadores e operandos encontrados. Quanto maior a extensão, maior a possibilidade de haver erros no código.

- Vocabulário do Programa

É a métrica obtida por meio da soma entre operandos e operadores distintos.

- Volume do Programa

É dado a partir do comprimento e do vocabulário. Quanto maior este número mais difícil se torna a manutenibilidade

- Dificuldade

A medida indica a dificuldade de codificação, quanto mais difícil a codificação, mais difícil a manutenção.

- Esforço

O esforço é o produto da Dificuldade pelo Volume, quanto mais alto está métrica, maior a dificuldade de manutenção.

#### 4.2.1.2 Complexidade Ciclomática de McCabe

Esta complexidade foi criada em 1976 por Thomas McCabe, e tem como objetivo medir o número de caminhos independentes de um módulo (FERREIRA, 2012).

Figura 3 – Fórmula de McCabe.

---


$$CC = E - N + 2p$$


---

Fonte: Ferreira (2012).

Onde E - Número de Arestas do Grafo; N - Número de Nós do grafo; p - Número de componentes conectados (FERREIRA, 2012). Quanto mais alto a complexidade ciclomática, mais alto a a manutenibilidade devido a complexidade em si.

#### 4.2.1.3 Maintainability Index

O Índice de manutenibilidade foi desenvolvido em 1994 por Oman. Esta métrica tem o objetivo de mensurar a manutenibilidade do software auxiliando no processo de manutenção, além de utilizar das medidas de Halstead e a Complexidade Ciclomática de McCabe (FERREIRA, 2012). Segundo Ferreira (2012) as medidas são descritas como:

Figura 4 – Índice de Manutenibilidade.

---


$$171 - (5,2 * \ln(\text{aveV})) - (0,23 * \text{aveV}(g')) - (16,2 * \ln(\text{aveLOC})) +$$

$$(50 * \sin(\sqrt{2,4\text{perCM}}))$$


---

Fonte: Ferreira (2012).

- **aveV** é a média por módulo da medida de Halstead;



- **aveLOC** a média de linhas de código por módulo;
- **aveV(g')** a média estendida da complexidade ciclomática de McCabe por módulo;
- **perCM** Média do percentual de comentários por linha de código. (Medida Opcional)

## 4.3 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).

### 4.3.1 Plano de Gerenciamento de Configuração

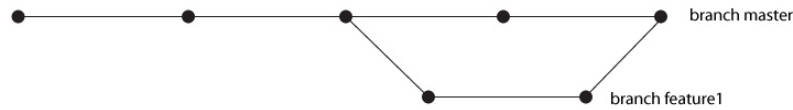
O Plano de Gerenciamento de Configuração (PGC) descreve todas as atividades de configuração e mudança que serão realizadas durante o projeto. Um conjunto de atividades, responsabilidades, ferramentas, recursos e etc. A gerência de configuração tem como objetivo garantir a integridade dos itens de configuração, que são qualquer artefato que esteja sob custódia da Gerência de Configuração, através do versionamento, da identificação, controlando mudanças e acesso.

### 4.3.2 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 especiais faz-se necessário que os desenvolvedores trabalhem em uma 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 5 demonstra a criação de um *branch* paralelo a linha de

Figura 5 – Branch no Sistema de Controle de Versão.



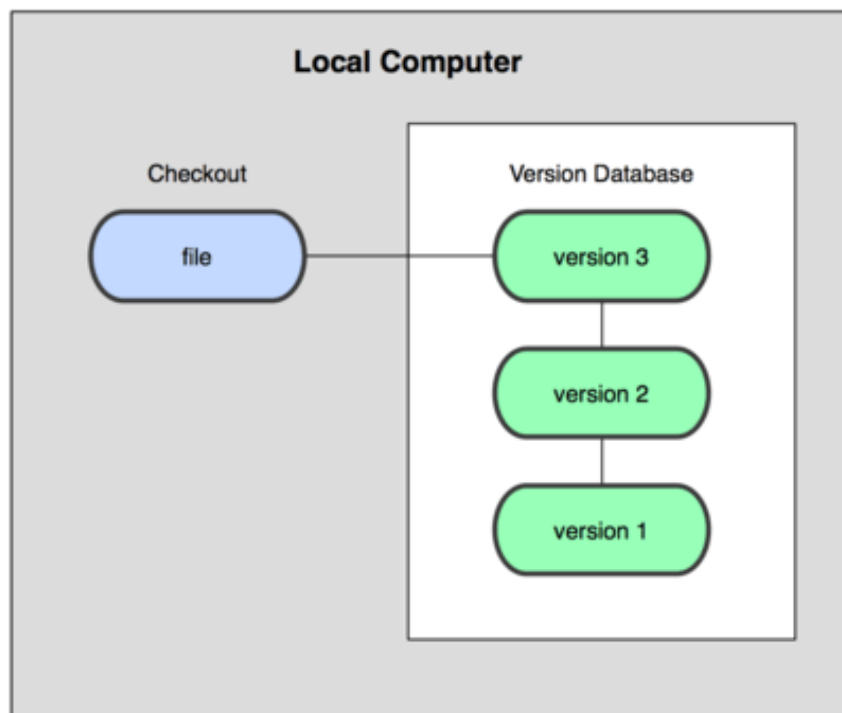
Fonte: Eis (2012).

desenvolvimento principal chamada de *branch feature1* e *branch master* respectivamente, posteriormente as ações realizadas no *branch feature1* são incorporadas ao *branch master*.

#### 4.3.2.1 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 6.

Figura 6 – Sistema de Controle de Versão Local.

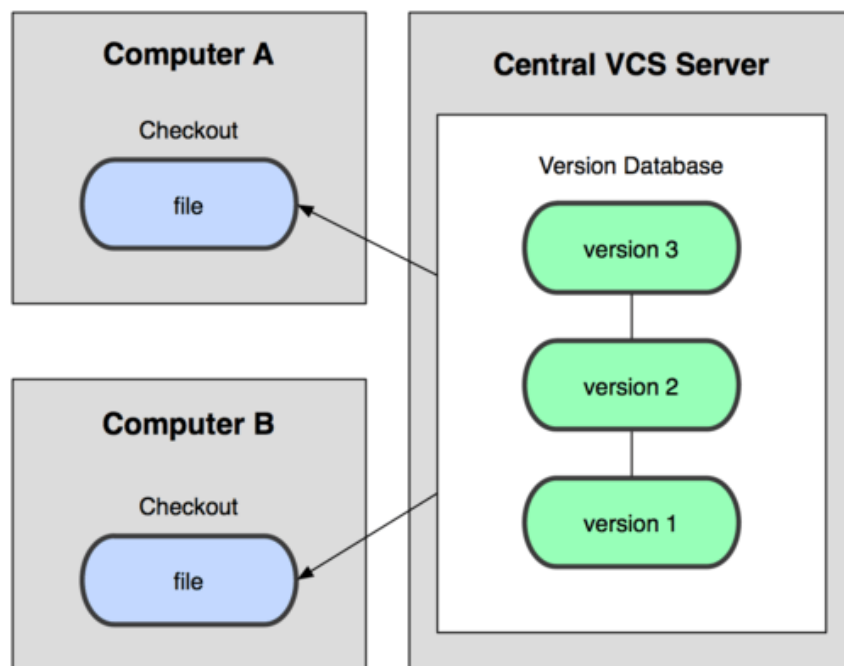


Fonte:(GIT, 20\_\_).

#### 4.3.2.2 Sistema de Controle de Versão Centralizado

Sistema de controle de versão centralizado como o nome diz possuem um único servidor centralizado, como o *subversion*<sup>1</sup>, *perforce*<sup>2</sup>, 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

Figura 7 – Sistema de Controle de Versão Centralizado.



Fonte: GIT (20\_).

trabalhando em máquinas diferentes realizam a comunicação com o servidor central para obter o artefato de trabalho.

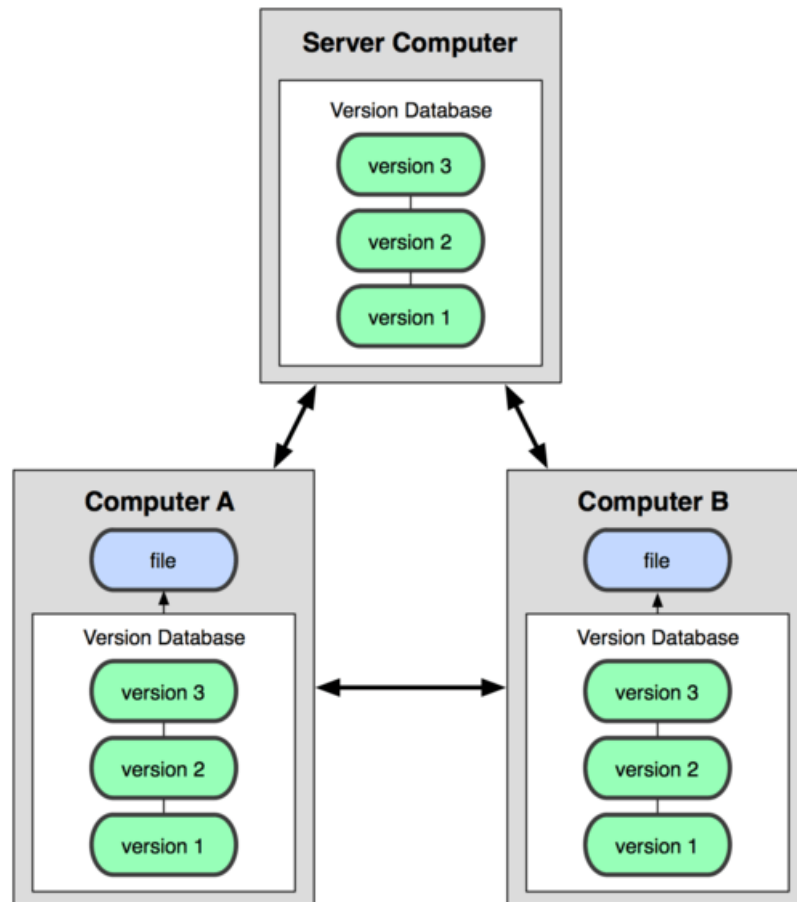
#### 4.3.2.3 Sistema de Controle de Versão Distribuído

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 8.

<sup>1</sup> <http://subversion.apache.org>

<sup>2</sup> <http://www.perforce.com>

Figura 8 – Sistema de Controle de Versão Distribuído.



Fonte: GIT (20\_\_).

### 4.3.3 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 mudanças são: *redmine*<sup>3</sup>, *GitHub*<sup>4</sup> *Jira*<sup>5</sup>

#### 4.3.4 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.

#### 4.3.5 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 9 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.

### 4.4 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:

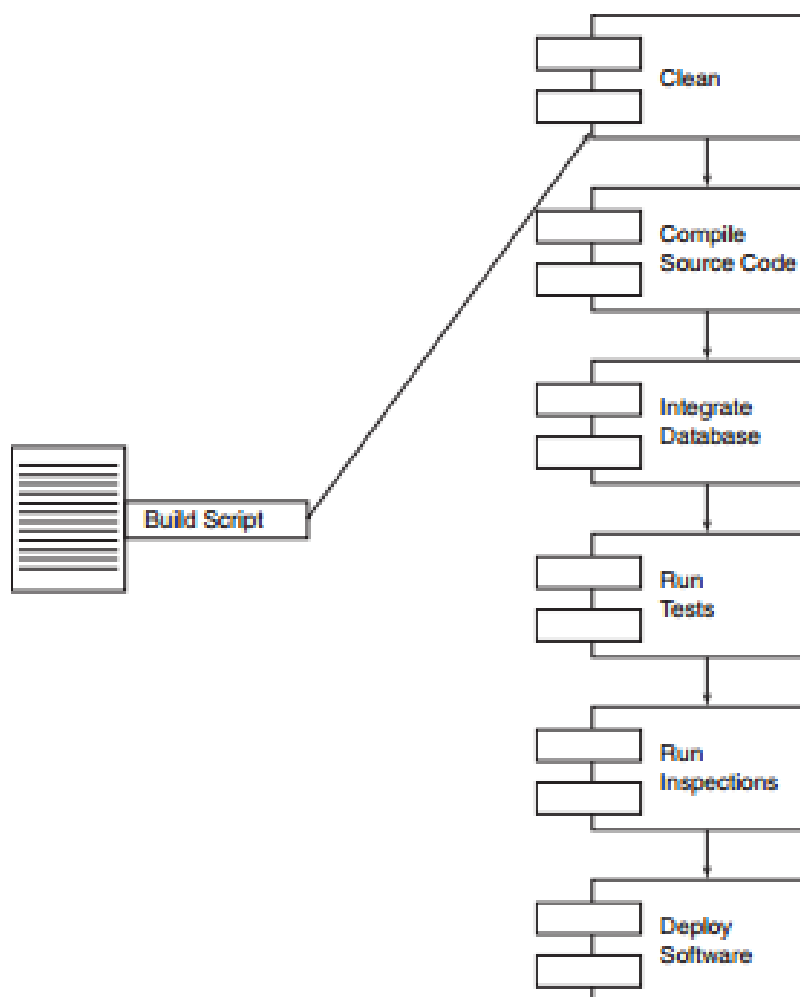
"[...] uma prática de desenvolvimento de software onde os membros de um time integram seu trabalho frequentemente, geralmente cada pessoa integra pelo menos diariamente – podendo haver múltiplas integrações por dia. Cada integração é verificada por uma build automatizada (incluindo testes) para detectar erros de integração o mais rápido possível. Muitos

<sup>3</sup> <http://www.redmine.org>

<sup>4</sup> <http://www.github.com>

<sup>5</sup> <https://www.atlassian.com/software/jira>

Figura 9 – Processo Lógico de uma Build.



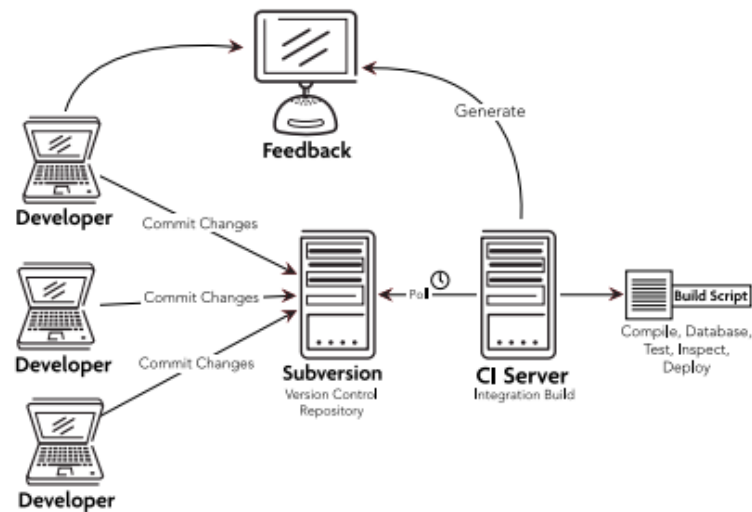
Fonte: Duval, Matyas e Glover (2007).

times acham que essa abordagem leva a uma significativa redução nos problemas de integração e permite que um time desenvolva software coeso mais rapidamente."Fowler (2000, tradução nossa)

A figura Figura 10 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:

Figura 10 – Ambiente de Integração Contínua.



Fonte: Duval (20\_\_).

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

#### 4.4.1 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.

#### 4.4.2 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).

#### 4.4.3 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 executa sua build privada com o objetivo de verificar possíveis erros gerados por essas modificações. Ver seção 4.4 Figura 10

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 que irão permitir concluir os objetivos definidos anteriormente.

O presente trabalho possui as seguintes características:

- **Pesquisa Aplicada:** Demonstra a utilização de métricas de software sobre o objeto em estudo;
- **Pesquisa Explicativa:** Explicita práticas utilizadas em um ambiente de integração contínua e analisa como estas práticas influenciam a manutenibilidade do software;
- **Pesquisa Quanti-Qualitativa:** Caracteriza-se como quantitativa pois traduz em números as informações obtidas para serem analisadas, mais precisamente a taxa de manutenibilidade. Qualitativa pois visa melhorar um atributo de qualidade, a manutenibilidade.
- 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 o Jenkins <sup>1</sup> 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 do 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 Ferreira (2012).

- Implementar a utilização da ferramenta e coletar resultados.

---

<sup>1</sup> <http://jenkins-ci.org/>

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. Os dados serão obtidos através da aplicação da fórmula de Manutenibilidade previamente definida antes e após a implantação da ferramenta de integração contínua, e os dados obtidos por meio da fórmula, serão confrontados a fim de fornecer os resultados desejados.

Esta sessão está definindo o cronograma das atividades realizadas neste projeto. Desde sua concepção até sua conclusão.

## 5.1 Cronograma de Execução

Tabela 1 – Cronograma das atividades previstas

ATIVIDADES	2014						
	Mai	Jun	Jul	Ago	Set	Out	Nov
Estudo de Campo	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, 20\_\_\_. Citado na página 22.

DUVAL, P. M.; MATYAS, S.; GLOVER, A. *Continuous Integration: Improving Software Quality and Reducing Risk*. 1. ed. [S.l.]: Pearson Education, 2007. Citado 2 vezes nas páginas 7 e 21.

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 17.

FERREIRA, I. A. *ANÁLISE DO IMPACTO DA APLICAÇÃO DE PADRÕES DE PROJETO NA MANUTENIBILIDADE DE UM SISTEMA ORIENTADO A OBJETOS*. 78 p. — Universidade Federal de Lavras, Lavras, 2012. Citado 6 vezes nas páginas 8, 12, 13, 14, 15 e 24.

FOWLER, M. Maio 2000. Disponível em: <<http://martinfowler.com/articles/continuousIntegration.html>>. Acesso em: 27.3.2014. Citado na página 21.

FURLANETO, R. *FERRAMENTA DE APOIO A GERÊNCIA DE CONFIGURAÇÃO DE SOFTWARE*. Dissertação (Mestrado) — Universidade Regional de Blumenau, Blumenau, Dezembro 2006. Citado na página 7.

GIT. 20\_\_\_. Disponível em: <<http://git-scm.com/book/pt-br/Primeiros-passos-Sobre-Controle-de-Vers~ao>>. Acesso em: 27.3.2014. Citado 3 vezes nas páginas 17, 18 e 19.

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 6 e 24.

KICHENBRAND, F. T.; FONTES, C. S. Marco 2012. Disponível em: <<http://rdribeiro.blogspot.com.br/2012/03/padroes-de-qualidade-isoiec-9126-nbr.html>>. Acesso em: 27.4.2014. Citado na página 13.

LIENTZ, B. P.; SWANSON, E. B. *Software Maintenance Management*. Reading, MA, USA: Addison-Wesley, 1980. Citado na página 10.

MENEZES, G. G. L. de. *OURIÇO: UMA ABORDAGEM PARA MANUTENÇÃO DA CONSISTÊNCIA EM REPOSITÓRIOS DE GERÊNCIA DE CONFIGURAÇÃO*. Niterói: [s.n.], 2011. Citado 3 vezes nas páginas 16, 22 e 23.

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 7.

PADUELLI, M. M. *Manutenção de Software: problemas típicos e diretrizes para uma disciplina específica*. São Carlos: [s.n.], 2007. Citado 2 vezes nas páginas 10 e 11.

- PADUELLI, M. M.; SANCHES, R. Problemas em manutenção de software caracterização e evolução. *III Workshop de manutenção moderna*, 2006. Citado na página 6.
- PFLEEGER, S. L. *Software Engineering: Theory and Practice*. New Jersey, USA: Prentice Hall, 2001. Citado 2 vezes nas páginas 10 e 11.
- PIGOSKI, T. M. *Practical Software Maintenance—Best Practices for Managing Your Software Investment*. [S.l.]: Katherine Schowalter, 1997. Citado 2 vezes nas páginas 10 e 11.
- PRESSMAN, R. S. *Engenharia de Software*. [S.l.]: Pearson Education, 2010. 1056 p. Citado 5 vezes nas páginas 6, 10, 16, 19 e 20.
- SOFTTEX. *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](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 16.
- SOMMERVILLE, I. *Engenharia de Software*. 9. ed. [S.l.]: Pearson Education, 2011. 529 p. Citado 2 vezes nas páginas 6 e 19.
- SOUZA, S. C. B. de et al. Documentação essencial para manutenção de software ii. *Workshop de Manutenção de Software Moderna*, n. 1, 2005. Citado 3 vezes nas páginas 10, 11 e 12.
- 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 8.