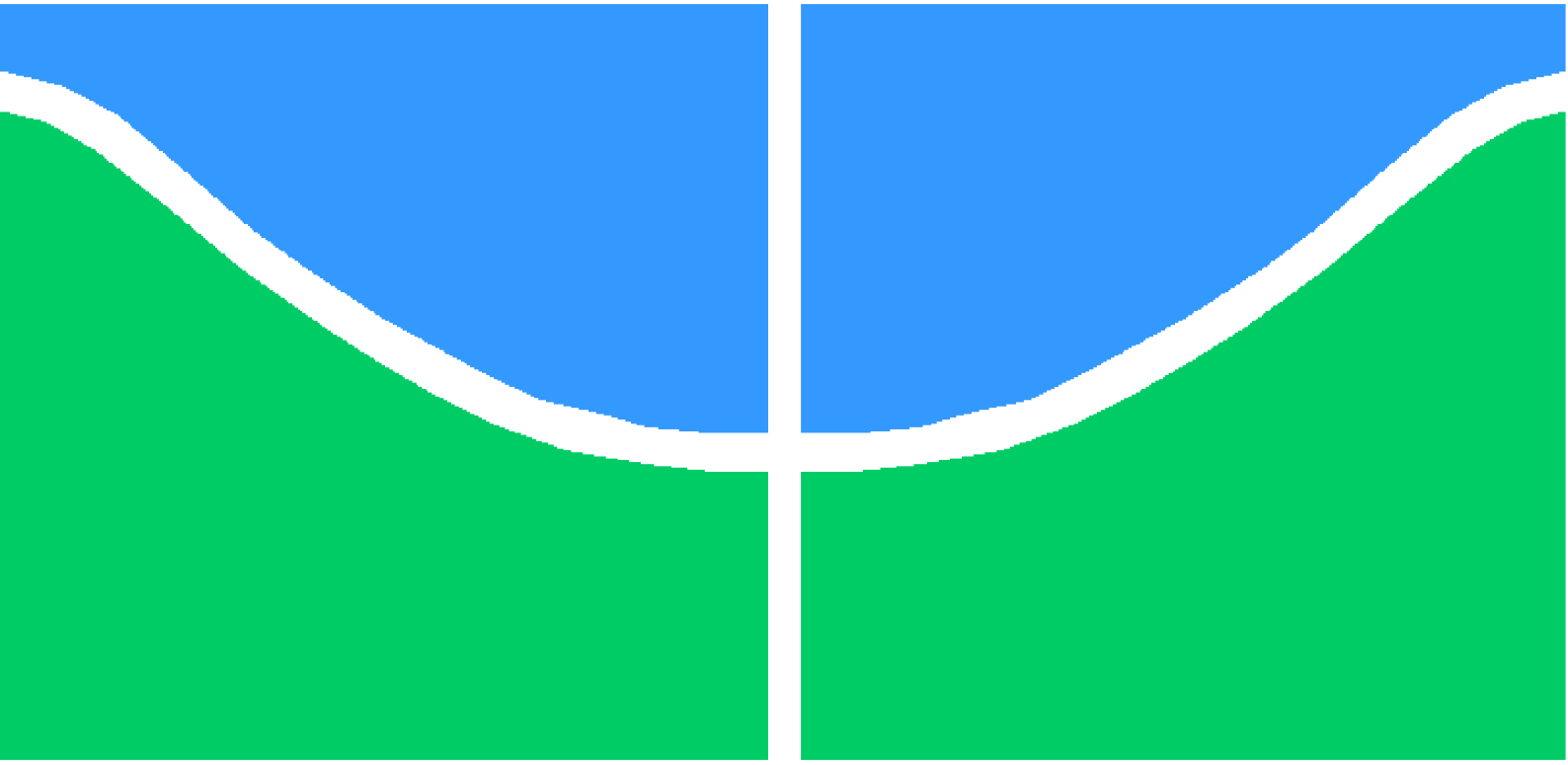

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Engenharia de Software

Criação de um Dashboard para Monitoramento de Perfis de Qualidade de Software

Autor: Levi Moraes dos Santos

Orientador: Prof. Dr. Maurício Serrano
Coorientadora: Profa. Dra. Milene Serrano

Brasília, DF

2016

Levi Moraes dos Santos

Criação de um Dashboard para Monitoramento de Perfis de Qualidade de Software

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software .

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: Prof. Dr. Maurício Serrano

Coorientador: Profa. Dra. Milene Serrano

Brasília, DF

2016

Levi Moraes dos Santos
Criação de um Dashboard para Monitoramento de Perfis de Qualidade de Software / Levi Moraes dos Santos. – Brasília, DF, 2016-
48 p. : il. ; 30 cm.

Orientador: Prof. Dr. Maurício Serrano

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2016.

1. Qualidade. 2. Dashboard. I. Prof. Dr. Maurício Serrano. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Criação de um Dashboard para Monitoramento de Perfis de Qualidade de Software

CDU 02:141:005.6

Levi Moraes dos Santos

Criação de um Dashboard para Monitoramento de Perfis de Qualidade de Software

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software .

Trabalho aprovado. Brasília, DF, 01 de junho de 2013:

Prof. Dr. Maurício Serrano
Orientador

Profa. Dra. Milene Serrano
Coorientadora

---

s
Convidado 1

Convidado 2

Brasília, DF
2016

•*Dedicatória ao Sr 1,2,3 de Oliveira 4*

Agradecimentos

Agradeço inicialmente a mim mesmo, porém agradecerei a mais pessoas quando me formar

Resumo

Palavras-chaves: qualidade. dashboard. monitoramento de métricas. visualização de métricas.

Abstract

This is the english abstract.

Key-words: latex. abntex. text editoration.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

MIT	Massachusetts Institute of Technology
ISO	International Organization for Standardization
APF	Administração Pública Federal
TCU	Tribunal de Contas da União
TI	Tecnologia da Informação
WDC	*Weighted Methods per Class*
DIT	*Depth of Inheritance*
NOC	*Number of Children*
CBO	*Coupling Between Objects*
RFC	*Response for a class*
LCOM	*Lack of Cohesion in Methods*

Sumário

1 Introdução

Neste primeiro capítulo é apresentado uma visão mais ampla do trabalho e que tem como objetivo introduzir a temática abordada. Este capítulo está dividido em 6 (seis) seções. Na primeira seção é abordado o contexto (Contextualização) em que se encontra este trabalho. Seguido da contextualização apresenta-se a problematica (Problema) na qual se deseja resolver com a solução apresentada. As justificativas (Justificativa) que levaram à realização deste trabalho. Após as justificativas são apresentados os objetivos (Objetivos) que servem como guia para solução proposta. Resultados esperados (Resultados Esperados) que como o próprio nome já sugere apresenta o que é esperado ao fim deste trabalho e por último organização do Trabalho (Organização do Trabalho) que descreverá o conteudo apresentado durante todo o trabalho.

1.1 Contextualização

O conceito de engenharia de software foi proposto inicialmente durante uma conferência na década de 60 em Garmisch na Alemanha. Nesta conferência estavam presentes usuarios, fabricantes e pesquisadores que debatiam sobre os constantes problemas no desenvolvimento de software [Paduelli 2007]. A qualidade na produção de software é uma área muito ampla e que abrange desde qualidade da arquitetura de software até qualidade no processo REFERENCIA. Este trabalho teve como objetivo central apresentar uma solução para visualização de métricas de qualidade dentro de alguns orgãos, tendo como os principais pilares três áreas comuns da engenharia de software, gerência de configuração, integração contínua e análise estática de código. Uma arquitetura próxima a essa vem sendo trabalhada dentro de alguns órgãos públicos do governo federal, entre eles o Tribunal de Contas da União o qual tem mostrado os melhores resultados nesta área. O problema encontrado atualmente está na falta de um acompanhamento na qualidade dos chamados softwares legados, softwares que são produzidos dentro/para o órgão e que passado um tempo ainda estão em atividade.

Dentro da área de Tecnologia as organizações tem encontrado um grande problema quando se trata de softwares legados. O trabalho feito por REFERENCIA prova que os softwares legados são esquecidos pelas organizações quando se fala sob uma perspectiva de manutenção. Contudo o estudo também revela que em grande parte das empresas esses mesmos softwares continuam rodando no ambiente de produção.

1.2 Problema de pesquisa

O principal produto da engenharia de software é o software, contudo o que tem se vivenciado na realidade brasileira de computação é que o software que está sendo entregue é um software precário e de baixa qualidade. Por ser uma palavra abstrata, o conceito de qualidade é bem amplo, porém o termo qualidade normalmente está associado a uma medida relativa, essa qualidade pode ser entendida como "conformidade às especificações". Conceituando dessa forma, a não conformidade às especificação é igual a ausência de qualidade.
Uma das grandes dificuldades nos orgãos públicos está no acompanhamento das manutenções prestadas por terceirizadas. Esse problema se agrava ainda mais quando a empresa contratante não consegue acompanhar ou não tem parametros concretos de indicadores de qualidade. Este trabalho tem como proposta a criação de uma dashboard de monitoramento para softwares legados onde é possível acompanhar de maneira simples e totalmente visual, indicadores de qualidade de código para projetos selecionados.
O grande desafio está em criar uma maneira de visualização de dados de maneira simplificada e objetiva para acompanhamento de software em um orgão público federal. Tomando este problema como base tem-se a seguinte questão de pesquisa:

"Uma vez definido os indicadores das métricas, como criar uma interface de visualização da informação? "

1.3 Justificativa

A motivação deste trabalho se deu como uma extensão de um trabalho de conclusão de concurso elaborado anteriormente em que eram tratados aspectos para monitoramento da qualidade dentro de um orgão público federal. O trabalho abordava aspectos de contratação de software dentro desses orgãos e como acontecia o acompanhamento desses softwares e propunha uma solução com integração contínua e gerência de configuração.
Após a conclusão do trabalho citado algumas lacunas continuaram, essas lacunas estão ligadas à apresentação destes dados para a equipe de gestão com o intuito de facilitar o acompanhamento das métricas.

1.4 Objetivos

1.4.1 Objetivos Gerais

Desenvolver uma solução intuitiva para monitoramento da qualidade de código de software

1.4.2 Objetivos Específicos

Para que seja possivel alcancar o objetivo geral alguns outros objetivos menores precisam ser alcancados para garantir o objetivo geral

- Identificar metricas de codigo ja existentes que mais se adequem as necessidades de um projeto de software
- Propor um ambiente integralizado e automatizado, englobando solucoes de analise estatica de código, integracao continua e versionamento de codigo
- Uma solução que agregue valor à equipe de manutenção

1.5 Resultados Esperados

1.6 Organização do trabalho

2 Referencial teórico

Este capítulo tem como objetivo servir como referencial teórico para todo o documento. As idéias discutidas neste capítulo são

2.1 Processo de Contratação de Software na APF

O Decreto n2.271 de 1997 [BRASIL 1997] dispõe sobre a contratação de serviços pela APF. Segundo o Decreto todos os produtos ou serviços que não apresentam relação direta com o propósito da instituição do governo federal devem ser tercerizados. O Decreto coloca como exemplo algumas atividades como por exemplo, conservação, limpeza, segurança, vigilância, transportes, informática, copeiragem, recepção, reprografia, telecomunicações são atividades livres para tercerização.
A licitação é um conjunto de processos administrativos de caráter formal para as compras de bens ou serviços nos governos federais, estaduais ou municipais. Segundo a Lei n 8.666 de 1993, o governo brasileiro para garantir a isonomia (princípio geral do direito segundo o qual todos são iguais perante a lei) diz que para contratações de bens ou serviços deve-se dar prioridade para licitação [BRASIL]. A licitação pode ocorrer de quatro categorias: concorrência, tomada de preços, convite e pregão [Brazil 2010].
Uma vez que uma empresa ganha a licitação para um serviço a empresa ganha o direito de contratação para prestar o serviço ao orgão contratante. Para ajudar no processo de contratação de empresas terceirizadas voltadas para a área de TI o TCU disponibiliza um Guia de boas práticas de contratações em soluções de TI [União 2012]. Segundo o guia a prática de contratação do serviço de desenvolvimento de um sistema de informação pode englobar elementos do tipo:

- Os softwares do sistema, devidamente documentados e com evidências de que foram testados;
- As bases de dados do sistema, devidamente documentadas;
- O sistema implantado no ambiente de produção do órgão;
- A tecnologia do sistema transferida para a equipe do órgão, que deve ocorrer ao longo de todo o contrato;
- As rotinas de produção do sistema, devidamente documentadas e implantadas no ambiente de produção do órgão;

- As minutas dos normativos que legitimem os atos praticados por intermédio do sistema;
- O sistema de indicadores de desempenho do sistema implantado, que pode incluir as atividades de coleta de dados para gerar os indicadores, fórmula de cálculo de cada indicador e forma de publicação dos indicadores. Citam-se, como exemplos, os indicadores de disponibilidade, de desempenho das transações e de satisfação dos usuários com o sistema de informação;
- Os scripts necessários para prover os atendimentos relativos ao sistema por parte da equipe de atendimento aos usuários, devidamente implantados e documentados;
- A capacitação dos diversos atores envolvidos com o sistema (e.g. equipe de suporte técnico do órgão, equipe de atendimento aos usuários, equipe da unidade gestora do sistema e usuários finais), que pode envolver treinamentos presenciais e a distância;
- O serviço contínuo de suporte técnico ao sistema (e.g. atendimento aos chamados feitos pelo órgão junto à contratada sobre dúvidas e problemas relativos ao sistema);
- O serviço contínuo de manutenção do sistema (e.g. implantação de manutenções corretivas e evolutivas).

2.2 Qualidade

O principal produto da engenharia de software é o software, contudo o que tem se vivenciado na realidade brasileira de computação é que o software que está sendo entregue é um software precário e de baixa qualidade. Por ser uma palavra abstrata, o conceito de qualidade é bem amplo, porém o termo qualidade normalmente está associado a uma medida relativa, essa qualidade pode ser entendida como "conformidade às especificações". Conceituando dessa forma, a não conformidade às especificação é igual a ausência de qualidade [Paduelli 2007].

A ISO 9126-1 proposta em 2001, também conhecida como Engenharia de Software - Qualidade do Produto, descreve o modelo de qualidade voltado para o produto de software como sendo composto por duas categorias como pode ser visto na Figura 1. A primeira categoria está relacionada a qualidade interna e a qualidade externa do software. A segunda categoria se relaciona com a qualidade de uso do software [NBR ISO/IEC 9126-1 2016]

Sob o aspecto de modelo de qualidade,a ISO 9126 classifica a qualidade interna do produto como sendo o somatório das características do ponto de vista interno do software. Os principais produtos desta categoria são os de cunho intermediário, entre eles: relatórios de analise estática do código fonte, revisão dos documentos produzidos, entre outros. A qualidade externa por sua vez já apresenta o seu foco mais voltado para as relações

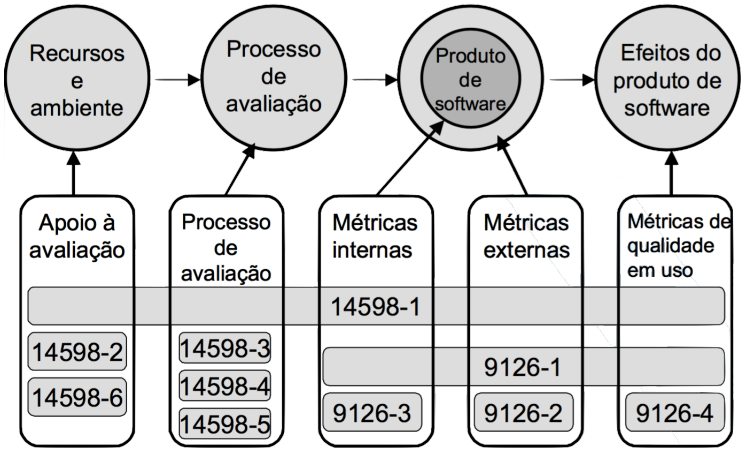

Figura 1 – Relação entre as NBR ISO/IEC 9126 e NBR ISO/IEC 14598 .Fonte: [NBR ISO/IEC 9126-1 2016]

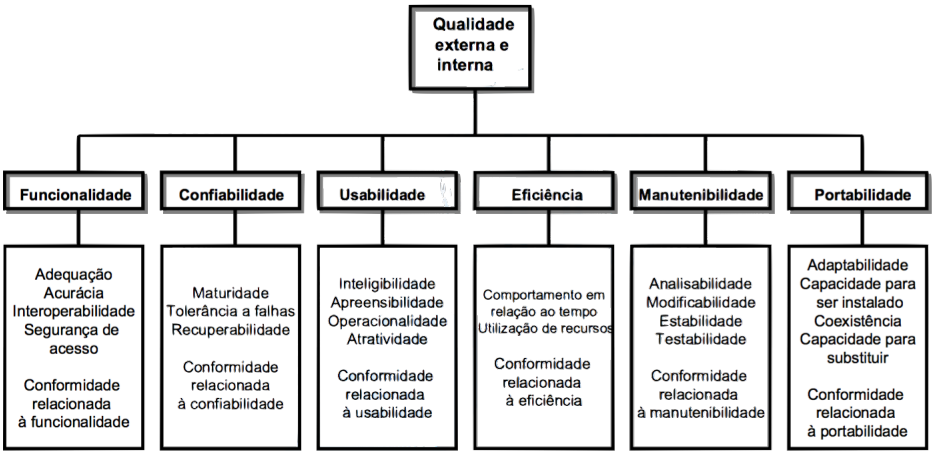

Figura 2 – Modelo de Qualidade para Qualidade Interna e Externa .Fonte: [NBR ISO/IEC 9126-1 2016]

externas do software, normalmente esta relacionado com a execução do código coletando suas métricas enquanto o software está em funcionamento. A Figura 2 apresenta a divisão proposta pela [NBR ISO/IEC 9126-1 2016] onde são categorizados seis aspectos de qualidade de software e suas subcaracterísticas, essas podem ser medidas por meio de métricas internas e externas.
Segundo a ISO 9126 essas caracteristicas podem ser definidas como:

- **Funcionalidade**: Capacidade do produto de software de prover funções que atendam às necessidades explícitas e implícitas, quando o software estiver sendo utilizado sob condições especificadas.

- **Confiabilidade**:Capacidade do produto de software de manter um nível de desempenho especificado, quando usado em condições especificadas.

- **Usabilidade**: Capacidade do produto de software de ser compreendido, aprendido, operado e atraente ao usuário, quando usado sob condições especificadas.
- **Eficiência**: Capacidade do produto de software de apresentar desempenho apropriado, relativo à quantidade de recursos usados, sob condições especificadas.
- **Manutenibilidade**: Capacidade do produto de software de ser modificado. As modificações podem incluir correções, melhorias ou adaptações do software devido a mudanças no ambiente e nos seus requisitos ou especificações funcionais.
- **Portabilidade**: Capacidade do produto de software de ser transferido de um ambiente para outro.

Em 2011 surgiu um conjunto de normas conhecidos como SQuaRE que traziam um framework aprimorado à atual norma vigente. Este framework tinha como objetivo avaliar o produto de qualidade de software.

2.2.1 Norma SQuaRE

O conjunto de normas SQuaRE (Requisitos e Avaliação de Qualidade de Sistema e Software) surgiu para substituir a ISO/IEC 9126. O objetivo destas normas é prover um framework que avalie a qualidade do produto de software [Schaidt 2014]. ISO/IEC 25010 mantém as características de qualidade já definidas na ISO 9126 com alguns incrementos.

- O escopo dos modelos de qualidade foram extendidos para incluir sistemas computacionais e a qualidade em uso pelo ponto de vista do sistema
- Segurança foi adicionada como característica, e não uma subcaracterística de funcionalidade.
- Compatibilidade foi adicionada como característica.
- A qualidade interna e externa foram combinadas como modelo de qualidade de produto.

A norma apresenta três guias de qualidade. O primeiro modelo é referente à Qualidade do Produto, o segundo da Qualidade em Uso e o último Qualidade de Dados.O modelo de Qualidade do Produto subdivide um sistema de software em oito categorias como mostra a imagem 3. Assim como a ISO 9126, a ISO 25010 também apresenta categorias, estas categorias se assemelham às categorias da ISO 9126 a qual serviu como base para criação da norma SQuaRE, estas caracteristicas são:

- **Adequação Funcional**: nível que determina o quanto um produto ou sistema satisfazem as especificações providas pelo usuário.

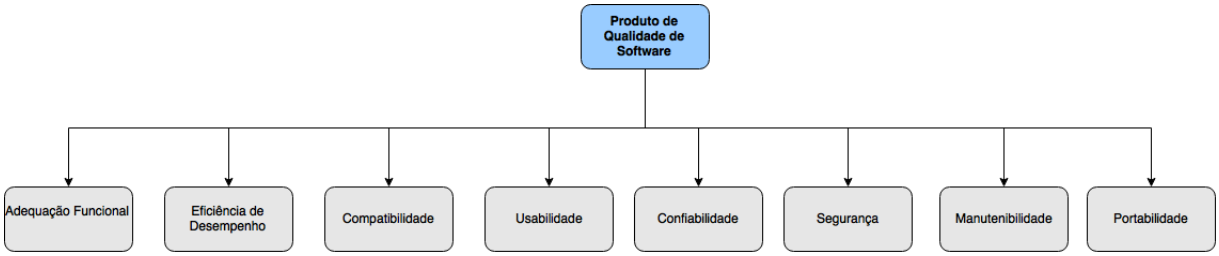

Figura 3 – Produto de Qualidade de Software.Fonte: [ISO 25010]

- **Eficiência de Performance**: performance relativa à quantidade de recursos usados em condições específicas.
- **Compatibilidade**: o nível que um sistema ou produto pode compartilhar informaçõesc com outros produtos, sistemas ou componentes.
- **Usabilidade**: O nível que um produto ou sistema pode ser usado por usuários específicos para atingir seus objetivos com efetividade, eficiência e satisfação em contexto específico de uso.
- **Confiabilidade**: nível que um sistema, produto ou componente executa suas atividades em um contexto pré-determinado e específico para uso.
- **Segurança**: nível no qual um sistema protege as informações e os dados de maneira que pessoas ou outros sistemas tenham acesso limitado de acordo com nível de autorização especifíco.
- **Manutenibilidade**: nível de efetividade e eficiência com o qual um produto ou sistema pode ser modificado pelos sistemas mantenedores.
- **Portabilidade**: Nível de efetividade e eficiência com o qual um sistema, produto ou componente pode ser transferido de um hardware, software ou ambiente de uso para outro.

Este trabalho tem seu desenvolvimento focado no modelo de Qualidade de Uso que apresenta características externas ao software e os resultados são coletados através de atributos estáticos [ISO 25010]. O foco deste trabalho está em medir indicadores quanto à manutenabilidade do software. Essa característica está diretamente ligada ao processo de Manutenção do Software.

2.2.2 Manutenção de Software

Segundo Sommervile [Sommervile 2011] manutenção de software é o processo de alterar o sistema depois que ele foi publicado. As alterações feitas no software podem ser

simples correções de erro, a até mudanças significativamente grandes que corrigem falhas arquiteturais, ou mesmo melhorias para acomodar novos requisitos.
Outra visão sobre manutenção de software é dada por Pressman [Pressman 2010] em que o autor conceitua o termo como sendo a correção de defeitos, adaptação do software para atender uma mudança do ambiente e aperfeiçoar as funcionalidades para que atendam às necessidades dos usuários. Outra característica do processo de manutenção é a sua composição por um conjunto de sub processos, atividades e tarefas que podem ser utilizados durante a fase de manutenção para alterar um produto de software, contanto que seja mantido o seu funcionamento [Calazans e Oliveira 2005].
Para Sommervile existem quatro categorias de manutenção:

- **Manutenção Corretiva**: seu objetivo está em identificar e remover falhas de software
- **Manutenção Adaptativa**: provê modificações no software para alojar mudanças no ambiente externo. Nesta manutenção também está incluso o processo de migração para diferentes plataformas tanto de software quanto de hardware.
- **Manutenção Perfectiva**: está manutenção é feito com o intuito de aperfeiçoar o software, além dos requisitos funcionais originais. Esta expansão dos requisitos traz consigo uma melhoria às funcionalidades até então implementadas ou um ganho de performance do sistema.
- **Manutenção Preventiva**: implementada para permitir que seja mais simples a correção, adaptação ou melhoria do software.

O modelo da figura 4 apresenta as atividades propostas por Pfleeger para um processo de manutenção. Na figura percebe-se que o processo de acompanhamento da manutenção ocorre durante todo o processo. As atividades apresentadas no diagrama são:

- **Analise do Impacto da Mudanção de Software**: estima o impacto de uma determinada mudança. Nesta atividade determina-se o grau de mudança e o quanto está mudança impactará no resto do software.
- **Entendimento do Software a ser Alterado**: nesta atividade são analisados os códigos-fonte do software para entender a mudança e a integração do que deve ser alterado. Esta atividade depende muito do grau de manutenabilidade do software, uma vez que quanto mais manutenível mais fácil e rapido se dá o processo de analise do software.
- **Implementação da Mudança**: incremento ou modificação do software. Está atividade é diretamente relacionada com o grau de adaptação do software, o quanto o

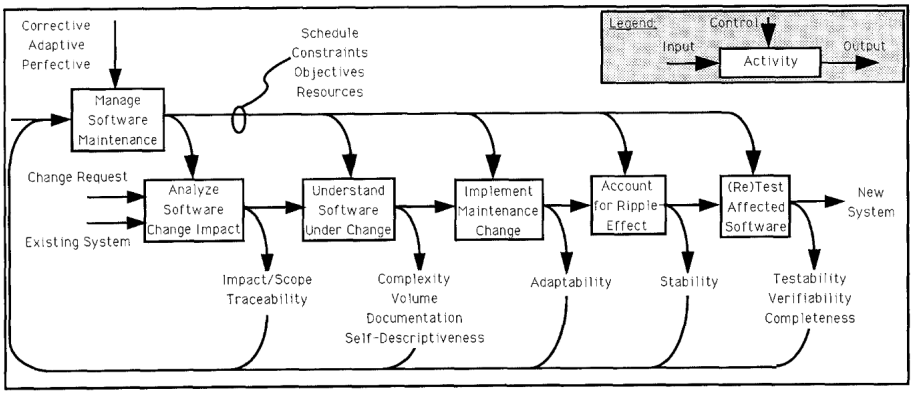

Figura 4 – Diagrama das Atividades de Manutenção de Software.Fonte: [Pfleeger e Bohner 1990]

software pode ser expandido ou comprimido. Essa característica de adaptabilidade é uma subcaracterística da Manutenabilidade de software apresentada pela norma Square.

- **Mudanças pelo Efeito Cascata**: Analise da propagação das mudanças ao longo do software. Essa atividade está intimamente relacionada com o indicador de coesão e acomplamento do software, este afere o quão amarrado estão as classes e os métodos do software.

- **(Re)Teste do Software**: é a ultima atividade antes da entrega do software alterado. O software é testado novamente sob a perspectiva do novo requisito.

Um estudo realizado por Kustyers e Heemstra [Kusters e Heemstra 2001] mostram a dificuldade de atuais na manutenção de software em seis grandes organizações da Alemanha. Um dos resultados obtidos foi que existe um uma falta muito grande na percepção quanto ao tamanho e o custo das manutenções de software. Os autores relatam que os gastos com manutenção são altos e que das seis empresas apenas uma mantinha registrado os seus processos de manutenção e os usava para fazer um novo planejamento. Normalmente tem se como verdade de que a manutenção de software está unicamente ligada ao conserto de *bugs*, entretanto estudos e *surveys* ao longo dos anos comprovam que mais de 80% do esforço gasto na manutenção é utilizado em ações não corretivas segundo Pigosky [Pigoski 1996]. O autor também afirma que entre 40% a 60% do esforço de manutenção está em entender o software que será modificado.

2.3 Métricas de Qualidade de Software

Uma métrica é uma função que pode ser medida, e métrica de qualidade de software é uma funcão na cuja entrada é uma informação de software e cuja saida é um único

valor numérico que pode ser interpretado como nível que é dado a um software [Kaner 2004]. Segundo [Pressman 2010] métricas podem ser definidas como sendo um pequeno subconjunto de informações que tem informações úteis acerca do software. Segundo Mills algumas características são inerentes a uma boa métrica, caracteristicas como, simplicidade,objetividade, fácil obtenção, validade, robustez, linearidade de escala. Diversos autores sugeriram conjunto de métricas que combinadas tratam de várias áreas de qualidade de software [Meirelles 2008].

2.3.1 Suíte de Chidamber-Kemerer

Este conjunto de métricas foi proposto por Shyam R. Chidamber e por Chris F. Kemerer em 1994 tem como objetivo avaliar aspectos de qualidade interna dos artefatos produzidos sob a visão de uma linguagem orientada a objetos [Chidamber 1994]. A suíte apresenta as seguintes métricas:

- **WMC**: Em uma classe com n métodos, a complexidade é dada como sendo a complexidade dos n métodos.

$$WMC = \sum_{i=1}^{n} Ci \tag{2.1}$$

  Essa métrica serve como indicador para o nível geral da modularização. Quanto maior o valor mais complexas estão a classe ou poucas classes possuem um indíce de complexidade muito alto.

- **DIT**: Tamanho do maior caminho entre a raiz da árvore de herança e a classe a qual está sendo analisada. Essa métrica permite ver o quanto uma mudança em uma determinada classe pode afetar todo o sistema.

- **NOC**: Quantidade de classes que se utilizam da classe em analise, seja por herança ou implementação. Essa métrica junto com outras ajuda a determinar quais classes são primordiais para o funcionamento do sistema.

- **CBO**: Está métrica é dada pela equação 2.2:

$$CBO = \frac{NumberOfDependencies}{NumberOfClassInPackage} \tag{2.2}$$

  Uma dependência pode ser definida como o uso de um método ou variável de outra classe porém do mesmo pacote.

- **RFC**: Número de métodos e construtores distintos que são chamados por uma classe. Está medida é muito utilizada para cobertura de testes, em que é pode ser constatado a necessidade ou não de uma modularização de uma classe.

- **LCOM**: Sendo C uma classe com n métodos, seja In o conjunto de variáveis de instância utilizadas pelo método n, seja P o conjunto tal que:

$$P = \{(Ii, Ij)|(Ii \bigcap Ij) = \varnothing\} \tag{2.3}$$

e Q o conjunto tal que:

$$Q = \{(Ii, Ij)|(Ii \bigcap Ij) = \varnothing\} \tag{2.4}$$

então LCOM é definida como sendo:

$$LCOM = |P| - |Q|se|P| > |Q| \tag{2.5}$$

ou zero caso contrário.

2.3.2 Suíte MOOD

Outro conjunto de métricas que tem como objetivo de medir de maneira quantitativa é a suíte de métricas MOOD. Ela conta com oito métricas. Essas métricas visam atender as principais características da orientação a objeto, então principios como polimorfismo, baixo acoplamento, encapsulamento e outros princípios são altamente valorizados. As métricas são [Meirelles 2008] [Moreira 2015]

- **MHF**: Métrica que indica a razão entre a soma de todos os métodos que são invisíveis em relação ao total de métodos do sistema.
  Número de métodos Visíveis em uma classe C

$$Mv(C) \tag{2.6}$$

Número de métodos encapsulados em uma classe C

$$Me(C) \tag{2.7}$$

O total de métodos é dado por

$$Mt(C) = Mv(C) + Me(C) \tag{2.8}$$

A equação que representa esta métrica é dada por:

$$MHF = \frac{\sum_{TC}^{i=1} Mh(Ci)}{\sum_{TC}^{i=1} Mt(Ci)} \tag{2.9}$$

Onde TC é o total de classes analisadas.

- **AHF**: Razão do somatório de todas os atributos que são herdados de todas as classes em relação ao número total de atributos. A equação que descreve este método é semelhante a dada por MHF

$$AHF = \frac{\sum_{TC}^{i=1} Ah(Ci)}{\sum_{TC}^{i=1} At(Ci)} \tag{2.10}$$

- **MIF**: Razão entre o somatório dos métodos herdados nas classes e o número total de métodos presentes no sistema.

$$MIF = \frac{TMh}{TMd} \tag{2.11}$$

- **AIF**: Razão entre a soma dos atributos herdados em todas as classes do sistema e o total de atributos da classe.

$$MIF = \frac{TAh}{TAd} \tag{2.12}$$

- **CFA**:Razão entre o total de acomplamentos permitídos no sistema e o atual número de acoplamentos possíveis por herança. Para está metrica toma-se como base uma relação cliente servidor entre as classes. Sempre que existir uma referência a um método ou atributo da classe servidora, usa-se a seguinte equação para calcular o fator de acoplamento.

$$COF = \frac{\sum_{TC}^{i=1}[\sum_{TC}^{i=1} isClient(Ci, Cj)]}{TC^2 - TC} \tag{2.13}$$

- **PFA**: Razão entre o numéro atual de possibilidades de polimorfismo diferentes que podem ser utilizados em uma classe e o número máximo de polimorfismos diferentes que podem haver nesta mesma classe.

Uma vez que as métricas se encontram definidas, deve-se pensar na melhor maneira de exibir as métricas para o usúario. O papel da visualização das métricas de software é definir com base na natureza, e na escala de uma métrica qual a melhor maneira de imprimir na tela essas informações.

2.4 Visualização da Informação

Computadores se tornaram peças fundamentais cotidiano do ser humano do século 21. Seja para lazer, estudo, comunicação, o computador revolucionou significamente em cada área que passou [Hasan e Abdul-Kareem 2014]. A visualização de software pode ser definida como uma disciplina que faz uso de várias formas de imagens que servem de insumo para compreender, entender e reduzir a complexidade dos sistemas de software existentes [Gračanin, Matković e Eltoweissy 2005]. Porém a visão que melhor se adapta ao contexto deste trabalho é dada por Gomes [Gomes e Tavares 2011] que diz que a visualização de uma forma generalizada é a construção de uma imagem visual na mente humana e está imagem vai além de representações gráficas ou conceitos.

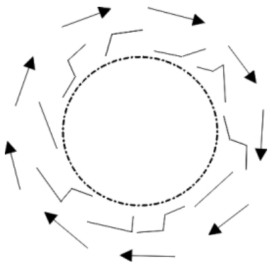

Figura 5 – Exemplo de glifo utilizado na visualização da variação das medidas relacionadas com o desenvolvimento computacional. Fonte: [Gomes e Tavares 2011]

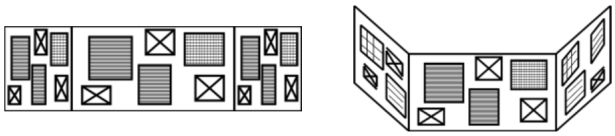

Figura 6 – Comparativo entre a)Bifocal Display e b) Perspective Wall

2.4.1 Visualização de Dados

Segundo Gomes [Gomes e Tavares 2011] algumas técnicas para visualização de dados são mais eficazes do que outras dependendo da natureza do dado a ser observado. Gomes destaca duas técnicas de visualização de dados.

- **Visualização de Atributos**: Nesta técnica a natureza do dado é representado através de um gráfico ou um mapa. Também é possível representar através de ícones ou *glifos*. ícones normalmente representam uma entidade ou um elemento que pertence a um contexto. O *glifo* representa um objeto geométrico que representa uma entidade ou um elemento do todo. O *glifo* tem uma forma que é definida pelo valor dos atributos da entidade como pode ser visto na figura 5.

- **Visualização de Estruturas e Relações**: está técnica utiliza do conjunto de dados ou elementos que podem ser distribuitos de maneira hierarquica. Dentro deste conceito existe uma ramificação para o uso da técnica aplicado de maneiras diferentes que possuem focos diferentes, algumas dessas técnicas são *Bifocal Display* em que a informação é apresentada em três seções distintas da minha representação, sendo a central a que contêm informações mais relevantes. Outra técnica utilizada é a *Perspective Wall* em que a informação é "projetada para uma parede". Uma comparação entre as duas técnicas pode ser observado na imagem 6.

Uma revisão sistemática feita com foco em visualização de software [Salameh, Ahmad e Aljammal 2016] apresenta alguns questionamentos levantados pelos autores referente ao objetivo de se utilizar a técnica. As respostas encontradas pelos autores estavam divididas

em quatro categorias. A primeira categoria encontrada baseava-se no artefato, a técnica era utilizada para acompanhar a evolução dos artefatos no repositório ao longo do tempo. A segunda categoria estava focada nas métricas, o objetivo estava em observar a alteração das métricas ao passar das entregas. A terceira categoria estava voltada para *features*, o acompanhamento focava em analisar como e quais *features* mudavam ao longo do tempo. A última categoria era centrada na arquitetura e em perceber as mudanças que estão envolvidas. Este trabalho tem seu objetivo focado na segunda categoria, acompanhar as métricas ao longo de um período de tempo e exibir para o usuário.
Para que se possa acompanhar as métricas dos projetos é necessário que se mantenha um histórico das métricas coletadas em momentos diferentes do desenvolvimento do software para que seja possível fazer uma comparação [Silva 2010]. Em seus estudos [Gračanin, Matković e Eltoweissy 2005] aponta três características que são necessárias para que se faça um bom acompanhamento histórico do software.

- **Tempo**: A visualização deve ser agrupada através do número da release ou outro indicador que possua um tempo definido. Deve ser feito um *snapshot* do sistema em que é possível ver claramente quais alterações foram realizadas.
- **Estrutura**: O sistema deve ser decomposto em sub-sistemas, cada sub-sistema decomposto em módulos e cada módulo responsável por uma parte do código.
- **Atributos**: Deve conter o número da versão, tamanho, alterações, complexidade.

Uma vez que é coletado os dados, eles ja estão categorizados de acordo com a sua natureza e ja podem ser exibidos ainda é necessário discutir qual a melhor forma de exibir todas essas informações na tela. Uma das formas de se agrupar toda a informação de maneira ordenada e com sentido lógico é através de *dashboards*. O conceito será apresntado no tópico a seguir

2.4.2 Dashboard

O conceito apresentado por Stephen Few em seu livro [Few 2006] diz que um *dashboard* é um *display* virtual das informações mais importantes para atingir um ou mais objetivos, ele deve ser construido e organizado para ser capaz de caber em uma única página para que a informação seja achada com facilidade.

3 Suporte Tecnológico

O objetivo desta seção é explicitar todo o aparato tecnológico a nível de software que foi utilizado durante o desenvolvimento deste trabalho. Esta seção está dividida em *Ferramentas para programação*.

3.1 Ferramentas para Programação

Neste tópico, serão apresentadas ferramentas e tecnologias voltadas ao contexto da Engenharia de Software que são utilizadas durante este trabalho, como, por exemplo, ferramentas para gerência de configuração e versionamento dos artefatos gerados.

3.1.1 GIT

A ferramenta GIT[1] foi desenvolvida por Linus Torvalds durante a criação do Kernel Linux, pois Linus percebeu que existia a necessidade de criar uma ferramenta *open-source* que fizesse o controle de versão [BENTO et al. 2013].
O motivo para escolha da ferramenta se deve ao fato de que o Git contém o suporte para desenvolvimento linear o que garante um paralelismo de diversas áreas do desenvolvimento. Outro diferencial do Git está nos *snapshots* dos objetos que são armazenados, isso significa que o Git não rearmazena arquivos que não foram alterados [MARTINHO e MUNIZ 2013].

3.1.2 Github

O Github[2] é um repositório *online* que fornece a criação de projetos públicos gratuitos. A ferramenta também provê um sistema de gestão para acompanhamento do desenvolvimento envolvendo um sistema de *logs*, gráficos de visualização e uma *Wiki* integrada a cada projeto [MARTINHO e MUNIZ 2013].

O principal motivo pela escolha do Github é que deseja-se disponibilizar a solução futuramente para consulta e aprimoramento de pessoas interessadas.

3.1.3 Bonita

A ferramenta Bonita[3] foi escolhida graças a sua facilidade de utilização e portabilidade para o sistema operacional Mac OS X e o fato de ser gratuita. Esta ferramenta

[1] https://git-scm.com/
[2] https://github.com
[3] http://www.bonitasoft.com/

auxilia na modelagem de processos.

3.1.4 Mac OS X

O sistema operacional Mac OS foi baseado no kernel Unix e fabricado e desenvolvido pela empresa Apple Inc. Utilizou-se a versão 10.11 do sistema também conhecida como "*El Capitan*".

3.1.5 LaTeX

O LaTeX[4] foi desenvolvido na década de 80 cujo objetivo era simplificar a diagramação de textos científicos e matemáticos, onde atualmente dispõe de uma grande quantidade de macros para bibliografia, referencias, gráficos entre outros.

3.1.6 Sublime Text 3

O Sublime Text 3[5] é um editor de texto bastante utilizado por programadores, por possuir apoio para diversas linguagens de programação, incluindo textos em LaTeX.

3.1.7 Zotero

Zotero [6] é um software para gerenciamento de referências bibliográficas. Ele possui integração com o *browser*, sincronização online e criação de bibliografias estilizadas.

[4] https://www.latex-project.org/
[5] https://www.sublimetext.com/3
[6] https://www.zotero.org

4 Proposta

5 Metodologia

5.1 Metodologia

A metodologia utilizada será a pesquisa exploratória utilizando da revisão sistemática para encontrar a resposta à seguinte pergunta: “ Uma vez definido os indicadores das métricas, como criar uma interface de visualização da informação ? ”. O uso da pesquisa exploratória se deu por conta da baixa produção de artigos na área. A revisão sistemática é uma técnica criada na área da medicina que se difundiu para outras áreas de conhecimento por causa da produtividade que se ganha ao se deparar com um conjunto de artigos que não se sabe quais que se adequam ao conteúdo do autor. A revisão sistemática se dará em três etapas consecutivas: planejamento, execução e analise.

Tabela 1 – Cronograma TCC 1

Cronograma	**Agosto**	**Setembro**	**Outubro**	**Novembro**
Realizar Pesquisa Bibliográfica	X	X	X	X
Estudar o orgão		X	X	
Propor Versão Inicial do dashboard			X	X

O plano metodológico consiste em duas fases iniciação e estudo da proposta. A primeira fase possui quatro atividades principais, são elas: Elaborar um roteiro de pesquisa, Pesquisar Referencias, Refinar Pesquisa e Catalogar material encontrado. Na atividade de elaboar um roteiro de pesquisa defini-se a string de busca e em quais bases serão pesquisados os materiais de referencia. A atividade de Pesquisar referencias como o próprio nome ja sugere envolve a pesquisa da string de busca nas bases selecionadas. Refinar pesquisa envolve elaborar uma nova string de busca que contenha termos mais especificos e que aumente a quantidade de arquivos referentes ao tema. Catalogar material é uma atividade focada em guardar os materiais encontrados colocando uma tag referente ao tema a que o artigo se refere e uma breve descrição sobre o que era mais importante.
A Segunda fase consiste em estudar o cenario em que será colocado a dashboard para que se possa entender quais as necessidades e os requisitos para implantação. Esta fase está dividida em três momentos: planejar, executar e checar. Durante o momento de planejar tem-se como principais atividades analisar o problema onde há um maior entendimento do contexto no qual será elaborado a solução. A segunda atividade é elaborar solução em que é esperado que ao fim dessa atividade exista um esboço do que será a solução final. No segundo momento implementa-se a solução e no terceiro momento acontece a validação

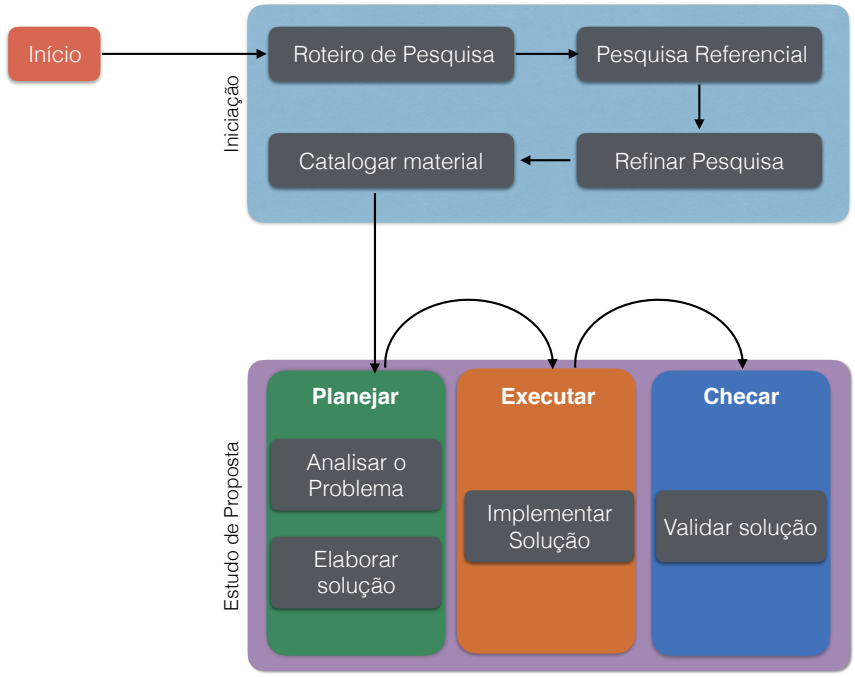

Figura 7 – Plano de Pesquisa

dessa solução.

6 Resultados Parciais

7 Considerações finais

Considero nada a ninguem

Referências

BENTO, G. V. et al. Análise Comparativa de Sistemas de Controle de Versões com foco em Versionamento de Banco de Dados Oracle. *e-RAC*, v. 3, n. 1, 2013. Disponível em: <http://www.computacao.unitri.edu.br/erac/index.php/e-rac/article/view/122>. Citado na página 37.

BRASIL. *Lei n 8.666, de 21 de junho de 1993*. Citado na página 25.

BRASIL. *DECRETO No 2.271, DE 7 DE JULHO DE 1997*. [S.l.], 1997. Citado na página 25.

Brazil (Ed.). *Licitações & contratos: orientações e jurisprudência do TCU*. 4a edição revista, ampliada e atualizada. ed. Brasília: TCU, Tribunal de Contas da União, 2010. ISBN 978-85-7018-319-4. Citado na página 25.

CALAZANS, A. T. S.; OLIVEIRA, M. A. L. Avaliação de Estimativa de Tamanho para projetos de Manutenção de Software. In: *Proc. of Argentine Symposium on Software Engineering*. [S.l.: s.n.], 2005. Citado na página 30.

CHIDAMBER, C. F. K. S. R. A metrics suite for object oriented design. In: *IEEE Transactions On Software Engineering*. [S.l.: s.n.], 1994. Citado na página 32.

FEW, S. *Information Dashboard Design - The effective Visual Comunication of Data*. [S.l.]: O'Reilly, 2006. Citado na página 36.

GOMES, L. F. O.; TAVARES, J. M. R. Percepção humana na visualização de grandes volumes de dados. In: *Actas do 10º Congresso Iberoamericano de Engenharia Mecânica (CIBEM 10)*. [s.n.], 2011. Disponível em: <https://repositorio-aberto.up.pt/handle/10216/56574>. Citado 3 vezes nas páginas 13, 34 y 35.

GRAČANIN, D.; MATKOVIĆ, K.; ELTOWEISSY, M. Software visualization. *Innovations in Systems and Software Engineering*, v. 1, n. 2, p. 221–230, set. 2005. Citado 2 vezes nas páginas 34 y 36.

HASAN, H.; ABDUL-KAREEM, S. Human–computer interaction using vision-based hand gesture recognition systems: a survey. *Neural Computing and Applications*, v. 25, n. 2, p. 251–261, ago. 2014. Citado na página 34.

ISO 25010. *Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models*. 2011. ed. [S.l.]. Citado 2 vezes nas páginas 13 y 29.

KANER, C. Software engineering metrics: What do they measure and how do we know? *10TH INTERNATIONAL SOFTWARE METRICS SYMPOSIUM*, 2004. Citado na página 32.

KUSTERS, R. J.; HEEMSTRA, F. J. Software maintenance: an approach towards control. In: *Software Maintenance, 2001. Proceedings. IEEE International Conference on*. [S.l.: s.n.], 2001. p. 667–670. ISSN 1063-6773. Citado na página 31.

MARTINHO, M.; MUNIZ, E. GIT SCM: Sistema de Controle de Versionamento Distribuído. *GIT SCM: Sistema de Controle de Versionamento Distribuído*, 2013. Disponível em: <http://www.tjam.jus.br/index.php?option=com_content&view=article&id=4032%3Agit-scm-sistema-de-controle-de-versionamento-distribuido&catid=344%3Asds-artigos-cientificos-e-tecnologicos&Itemid=455&showall=1>. Citado na página 37.

MEIRELLES, P. R. *Levantamento de Métricas de Avaliação de Projetos de Software Livre*. Dissertação (Mestrado) — Universidade de São Paulo, 2008. Citado 2 vezes nas páginas 32 y 33.

MOREIRA, L. N. Avaliação de qualidade de software baseada em métricas estáticas: um estudo de caso no Tribunal de Contas da União. 2015. Disponível em: <http://bdm.unb.br/handle/10483/10107>. Citado na página 33.

NBR ISO/IEC 9126-1. [S.l.], 2016. v. 5, n. 7, 088–108 p. Disponível em: <http://periodicos.udesc.br/index.php/reavi/article/view/2316419005072016088>. Citado 3 vezes nas páginas 13, 26 y 27.

PADUELLI, M. M. *Manutenção de Software: problemas típicos e diretrizes para uma disciplina específica*. Dissertação (Mestrado), 2007. Citado 2 vezes nas páginas 21 y 26.

PFLEEGER, S. L.; BOHNER, S. A. A framework for software maintenance metrics. In: *Software Maintenance, 1990, Proceedings., Conference on*. [S.l.]: IEEE, 1990. p. 320–327. Citado 2 vezes nas páginas 13 y 31.

PIGOSKI, T. M. Pratical software maintenance: Best practices for managing your software investment. *Wiley Computer Publishing*, 1996. Citado na página 31.

PRESSMAN, R. S. *Software Engineering - A Practioner's Approach*. 7. ed. [S.l.]: Higher Education, 2010. Citado 2 vezes nas páginas 30 y 32.

SALAMEH, H. B.; AHMAD, A.; ALJAMMAL, A. Software evolution visualization techniques and methods-a systematic review. In: *Computer Science and Information Technology (CSIT), 2016 7th International Conference on*. [S.l.]: IEEE, 2016. p. 1–6. Citado na página 35.

SCHAIDT, Y. R. L. *Monitoramento de Qualidade de Software na Administração Pública Federal*. Dissertação (Mestrado) — Universidade de Brasília, 2014. Citado na página 28.

SILVA, M. A. da. *IAVEMS: Infraestrutura de Apoio à Visualização da Evolução de Métricas de Software*. Tese (Doutorado) — Universidade Federal do Rio de Janeiro, 2010. Citado na página 36.

SOMMERVILE, I. *Software Engineering*. 9. ed. [S.l.]: Pearson, 2011. Citado na página 29.

UNIÃO, T. de Contas da. *Guia de boas práticas em contratação de soluções de tecnologia da Informação*. [S.l.], 2012. Citado na página 25.