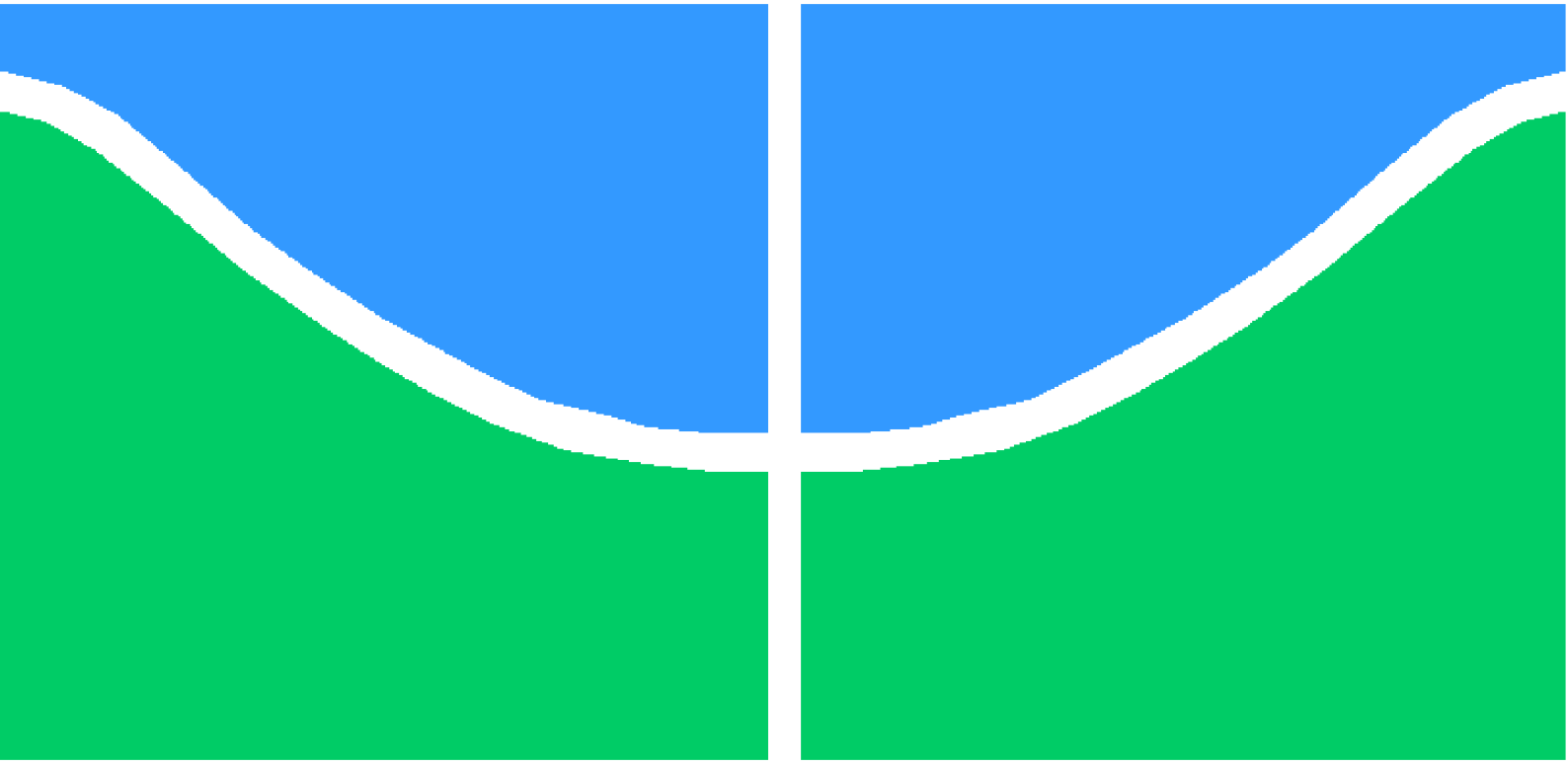

Universidade de Brasília - UnB

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-
64 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*
DNPM	Departamento Nacional de Produção Mineral
UI	*User Interface*
SCM	*Software Configuration Management*
API	*Application Programming Interface*
TCC	Trabalho de Conclusão de Curso

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]. Desde então empresas, orgãos públicos e diversas outras instituições utilizam do computador para automatizar tarefas ou cálculos antes feitos por humanos [Filho 2007]. O Decreto n2.271 de 1997 [BRASIL 1997] coloca a atividade de informática (e seus afins) como sendo um dos tipos de serviço passíveis de tercerização, ou seja, uma empresa tercerizada deve realizar os serviços referentes à este tipo de atividade. Uma vez que o software é produzido por uma empresa tercerizada é necessário que se faça o controle da qualidade deste software.
O controle da qualidade é um dos processos no ciclo de vida de desenvolvimento de software [Machado e Souza 2004]. A qualidade de software na produção do software é uma área muito ampla e que abrange desde qualidade da arquitetura de software até qualidade no processo . 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 softwares entregues pelas empresas tercerizadas

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 por Luiza e Yago [Schaidt 2014] 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 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.
Este trabalho também teve como base outro trabalho de conclusão de concurso de Adriano [Silva 2014], contudo o propósito deste trabalho é apresentar uma visão diferente a já abordada. O foco deste trabalho está em facilitar a comunicação entre a tercerizada e o gestor de projetos apresentando uma ferramenta que fará a ponte entre as duas partes.

1.4 Objetivos

1.4.1 Objetivos Gerais

Desenvolver uma solução 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

- A partir de um conjunto de métricas já estabelecidas, acompanhar a evolução de em um projeto de software.
- Utilizar dos dados de uma ferramenta de analise estática para coleta de métricas.
- Propor uma ferramenta automatizada de visualização e acompanhamento de métricas para um projeto.
- Verificar a qualidade de um software com base em editais de contratação.

1.5 Resultados Esperados

Ao fim deste trabalho espera-se apresentar uma solução em software para visualização de métricas coletadas juntamente com uma ferramenta de analíse estática, mostrar um *dashboard* que indica a atual situação de um projeto do ponto de vista de um orgão público.

1.6 Organização do trabalho

1.7 Resumo do Capítulo

2 Referencial teórico

Este capítulo tem como objetivo servir como referencial teórico para todo o documento. Inicialmente o capítulo apresenta uma visão geral sobre o processo de contratação de software na APF, destacam-se a importancia de algumas leis que servem como fundamento para a contratacao de terceirizadas e o nível de qualidade que é exigido por parte dos orgãos públicos. Em seguida discute-se sobre o conceito de qualidade de software na visão de alguns atores e o que é qualidade segundo a ISO 9126 que serviu como base para a norma SQUARE também discutida neste trabalho. Após a definição de qualidade pela norma SQUARE, é apresentado conceitos fundamentais de manutenção de software que servem como guia para o desenvolvimento deste trabalho. Tendo definido manutenabilidade alguns conjuntos de métricas que visam identificar possíveis lacunas de manutenabilidade no código são apresentados. Com as métricas definidas é necessário que se faça a devida visualização das métricas, tópico final deste capítulo.

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).

Tendo definido o que pode ser terceirizado dentro de um órgão público, é necessário que o produto entregue tenha qualidade aceitável para que seja fechado o acordo por parte da tercerizada e da contratante (neste caso APF). Contudo é necessário estipular o que é avaliado durante a contratação de um software.

2.1.1 Avaliação da Qualidade Em Um Processo de Contratação

Uma vez que o software é produzido pela tercerizada o software passa por um processo de verificação de todo o artefato que foi entregue seja ele em forma de código, como em forma de documentos. São lançados dezenas de editais todos os anos com o intuito de suprir a necessidade dos órgãos públicos quanto a demanda de software. Cada órgão é responsável por disponibilizar o seu edital contendo entre vários assuntos a forma

como será inspecionado o código entregue pela tercerizada. Em um edital feito pelo DNPM [Mineral 2015] um dos objetos de avaliação é a cobertura miníma de testes como mostra a Tabela 1, são avaliados tanto testes unitários, como de interface e de integração.

Tabela 1 – Índice de Cobertura Por Tipo de Teste do Edital da DNPM. Fonte: [Mineral 2015]

Tipo de Teste	**% de cobertura**
Unitários	70%
De Integração	100%
De Interface	20%

Este mesmo edital também apresenta outra tabela 2 mostrando outros detalhes que também serão cobrados na entrega do software. Algumas características desta tabela devem ser salientadas, como por exemplo, algumas métricas (mais especificamente taxa de cobertura de código, complexidade por método e LCOM4) não possuem valores definidos de cobrança e são ajustados de acordo com o projeto.

Tabela 2 – Métricas de Qualidade de Código Exigidas pela DNPM. Fonte: [Mineral 2015]

Métrica	**Meta**	**Severidade**
Taxa de cobertura de código	Definida na Demanda	Média
Complexidade por método	Definida na Demanda	Média
Coesão (LCOM4)	Definida na Demanda	Média
Violações do tipo Blocker	Zero	Média
Violações do tipo Critical	Zero	Média
Violações do tipo Major	Igual ou menor que 0,5% em relação ao total de linhas de código	Baixa
Violações do tipo Minor	Igual ou menor que 1% em relação ao total de linhas de código	Baixa
Taxa de sucesso em testes unitários	100%	Baixa
Taxa de duplicações de blocos	Igual ou menor que 2%	Baixa
Taxas de comentários da API Pública	Maior ou igual a 80%	Baixa
Linhas de código comentadas	Igual ou menor que 0,1% em relação ao total de linhas de código	Baixa

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]

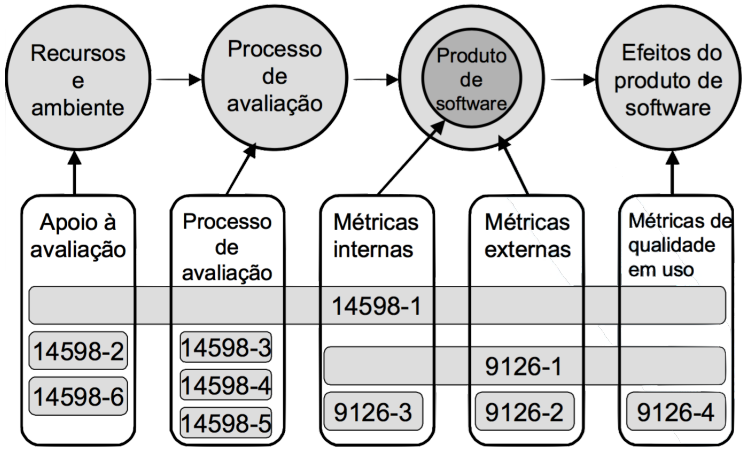

Figura 1 – Relação entre as NBR ISO/IEC 9126 e NBR ISO/IEC 14598 .Fonte: [NBR ISO/IEC 9126-1 2016]

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 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.

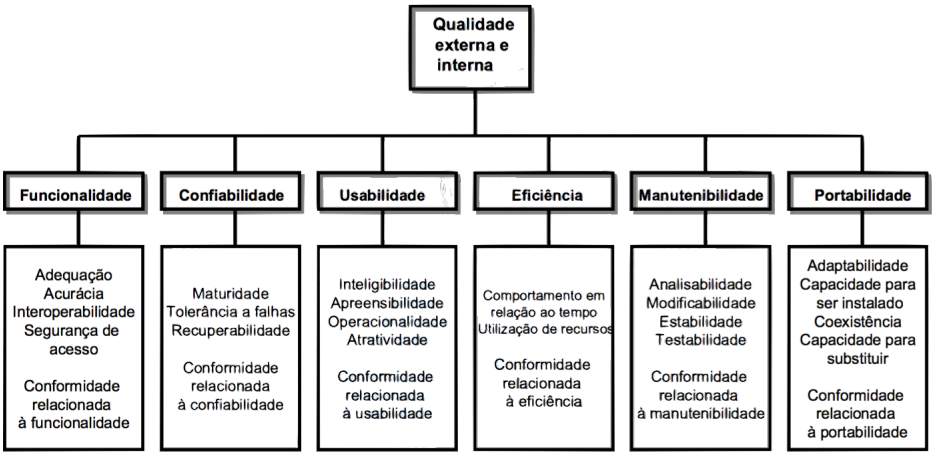

Figura 2 – Modelo de Qualidade para Qualidade Interna e Externa .Fonte: [NBR ISO/IEC 9126-1 2016]

- **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.
- **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.

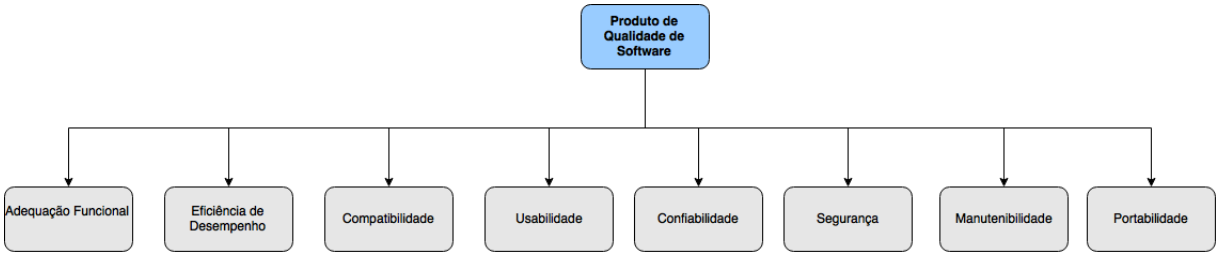

Figura 3 – Produto de Qualidade de Software.Fonte: [ISO 25010]

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 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.

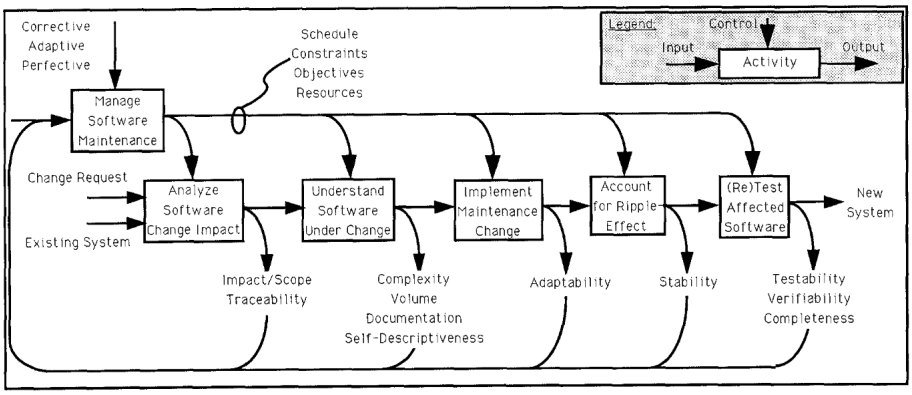

Figura 4 – Diagrama das Atividades de Manutenção de Software.Fonte: [Pfleeger e Bohner 1990]

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.

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.

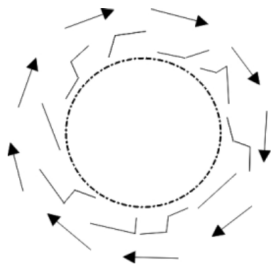

Figura 5 – Exemplo de glifo utilizado na visualização da variação das medidas relacionadas com o desenvolvimento computacional. Fonte: [Gomes e Tavares 2011]

- **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.

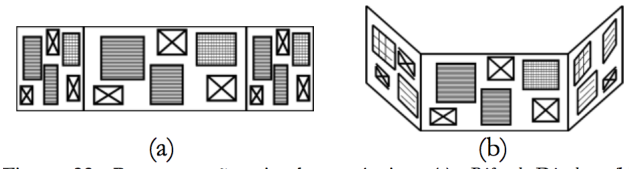

Figura 6 – Comparativo entre a)Bifocal Display e b) Perspective Wall

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 apresentado 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. Esta seção visa apresentar alguns modelos de *dashboards* e suas características.

2.5 Resumo do Capítulo

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 e Ferramentas de Gerenciamento .

3.1 Ferramentas de 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 SonarQube

O SonarQube é uma ferramenta *open-source* de analise estática de código-fonte que foca em analisar sete ramos da qualidade de código como pode ser visto na figura 7. A fer-

[1] https://git-scm.com/
[2] https://github.com

ramenta apresenta métricas quanto a duplicação de código, testes unitários, complexidade, *bugs* em potencial, regras da linguagem, comentários e arquitetura e design [SonarQube documentation]. A ferramenta foi escrita em Java e seu foco está em lidar com defeitos de código, contudo existe uma diversidade de plugins enorme que extende as funcionalidades da ferramenta [Ferenc et al. 2014]. A arquitetura do SonarQube é composta de quatro

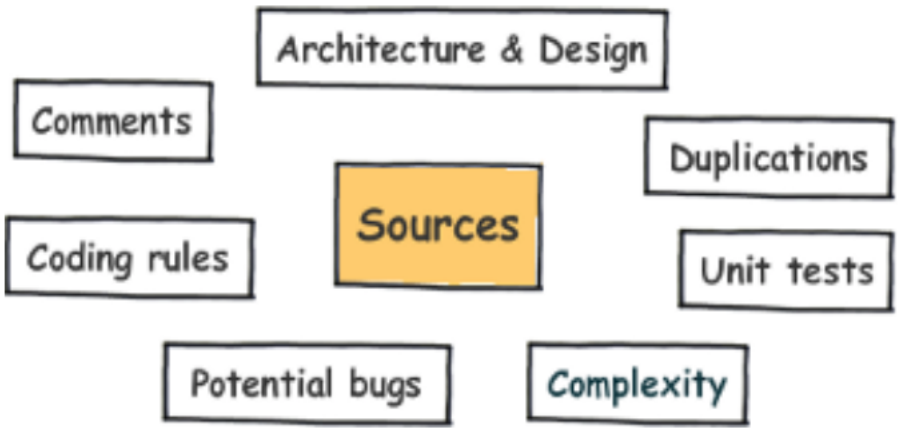

Figura 7 – Sete aspectos de qualidade de código cobertos pelo SonarQube. Fonte: [SonarQube documentation]

componentes como pode ser visto na figura 8 [SonarQube documentation].

1. ***SonarQube Server*** composto de três atividade principais:
   a) Um ***Web Server*** para os desenvolvedores, gerentes que procuram *snapshots* da qualidade do código, e configurar uma instancia do SonarQube.
   b) Um ***Search Server*** que se baseia no conceito de *Elasticsearch* para devolver as buscas para a UI.
   c) Um ***Compute Engine Search*** responsável pelo processamento de relatórios de análise de código e de salvá-los no banco de dados SonarQube.
2. Um **SonarQube Database** que armazena a configuração da instancia do SonarQube e os *snapshots* de qualidade dos projetos.
3. **Plugins** que são instalados no servidor, normalmente são plugins de linguagem, integração com outras ferramentas, autenticações entre outros.
4. Um ou mais ***SonarQube Scanners*** que rodam na *build* ou nos servidores de integração contínua para analisar projetos.

O SonarQube também é facilmente integrado com outras ferramentas utilizadas durante o ciclo de vida do software. O esquema da figura 9 apresenta a implantação do SonarQube em diversos estágios do desenvolvimento de software [SonarQube documentation].

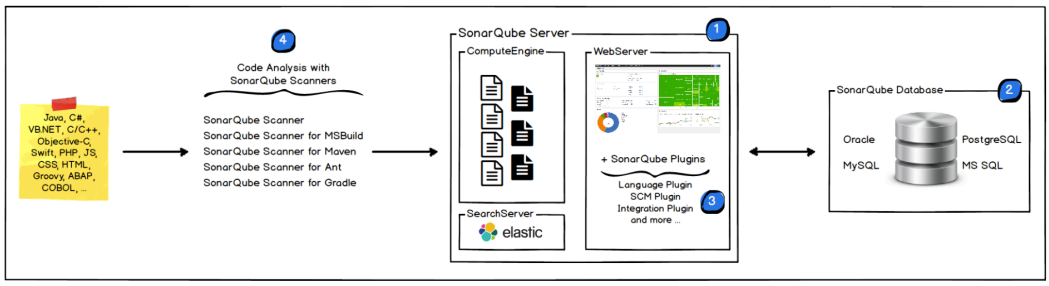

Figura 8 – Arquitetura SonarQube composta de quatro componentes: [SonarQube documentation]

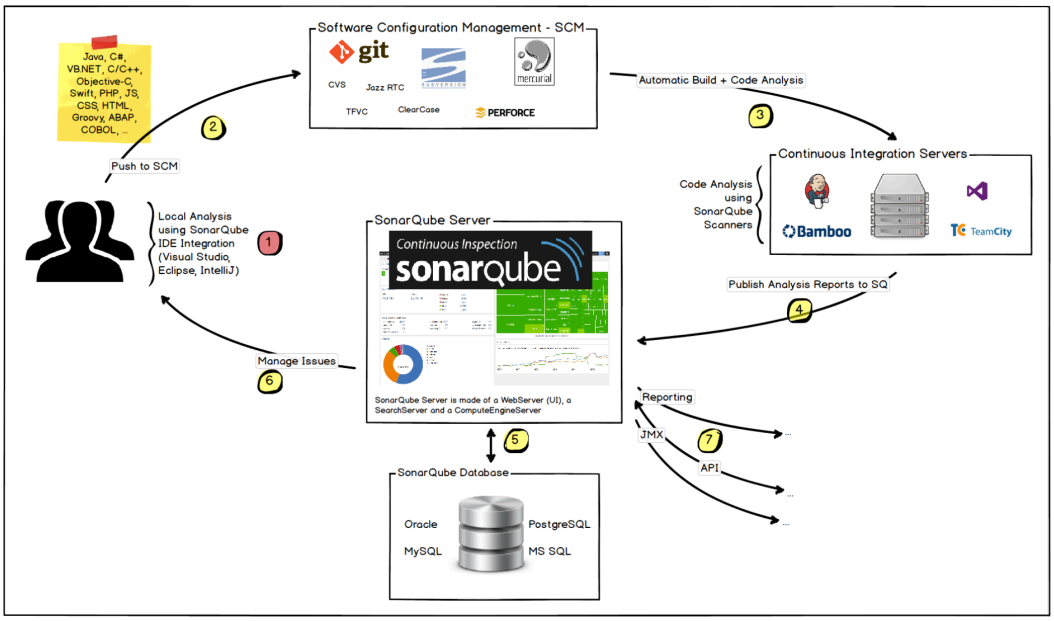

Figura 9 – Integração SonarQube em Diversas Áreas de Desenvolvimento de Software: [SonarQube documentation]

1. Os desenvolvedores podem utilizar uma instância do SonarQube (SonarLint) em suas próprias IDEs para rodar uma análise local.

2. Desenvolvedores podem subir seus códigos para suas ferramentas de SCM favoritas (Git, SVN).

3. O servidor de integração contínua desencadeia uma compilação automática, e a execução do SonarQube Scanner necessário para executar a análise de SonarQube.

4. O relatório de análise é enviado para o servidor do SonarQube para processamento.

5. SonarQube Server processa e armazena os resultados do relatório de análise do banco de dados SonarQube e exibe os resultados na interface do usuário.

6. Os desenvolvedores analisam, comentam, arruman seus problemas para gerenciar e reduzir sua dívida técnica através do SonarQube UI.

7. Gestores recebem um relatório da análise

A escolha do SonarQube como ferramenta de analise estática se deu pelo fato de que grande parte dos orgãos públicos utilizam esta ferramenta para fazer a avaliação da qualidade dos softwares entregues pelas tercerizadas. Muito dos editais encontrados neste trabalho utilizavam o SonarQube como uma das ferramentas de audição dos softwares entregues.

3.1.4 Google Charts e Charts.js

O *Google Charts* é uma API disponibilizada pelo Google, ela fornece uma maneira perfeita visualizar dados em um site. De gráficos de linha simples a complexa árvore hierárquica de mapas, o *google charts* fornece um grande número de tipos de gráficos prontos para uso [Google Charts documentation]. A forma mais comun de utilizar o *google charts* é utilizando um JavaScript dentro do código HTML. Os gráficos são visualizados utilizando classes do JavaScript e são renderizados através de HTML5/SVG o que garante o funcionamento em diversos *browsers*.
Outra API utilizada é a Charts.js que é uma API open-source que também utiliza a ferramenta JavaScript e renderiza os gráficos com HTML5. O principal motivo para se utilizar o Charts.JS juntamente com o Google Charts é que a API da google não fornece o gráfico de Radar o qual é utilizado neste trabalho [ChartsJS documentation].

3.2 Ferramentas de Gerenciamento

3.2.1 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 auxilia na modelagem de processos.

3.2.2 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] http://www.bonitasoft.com/

3.2.3 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.

3.2.4 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.

3.2.5 Zotero

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

3.3 Resumo do Capítulo

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

4 Proposta

O objetivo deste capítulo é apresentar uma possível proposta de uma solução que atenda às necessidades colocadas no primeiro capítulo referente a problemática deste trabalho. A proposta está dividida em duas partes. A primeira parte está relacionada à coleta das métricas utilizando a ferramenta SonarQube, e a segunda parte está relacionada na criação do *dashboard* e a visualização das informações. Para analise deste projeto foram escolhidos dois projetos, sendo o projeto PROJETO 1 e o PROJETO 2 de código aberto DO ORGAO como base para elaboração deste trabalho. A terceira e última parte do tarabalho consiste em analisar o trabalho realizado através de uma avaliação qualitativa de possíveis usuários.

4.1 Coleta das Métricas

Para fazer a coleta das métricas vai ser utilizado a ferramenta SonarQube que como já dito na seção de Suporte Tecnológico é muito utilizado em órgãos públicos e em editais por ser uma ferramenta *open-source*. A coleta das métricas é feita com base no edital feito pelo DNPM [Mineral 2015] que se baseia na Suíte de Métricas de Chidamber-Kemerer. As métricas e suas metas estão apresentadas a seguir.

- Taxa de Cobertura de Teste Unitário = 80;
- LCOM 4 = 10;
- Violações do tipo Blocker = 0;
- Violações do tipo Critical = 0;
- Violações do Tipo Major = 0,5 total de linhas de código;
- Violações do Tipo Minor = 1 total de linhas de código;
- Taxa de Duplicações de blocos = igual ou menor que 2 total de linhas de código;
- Linhas de Código Comentadas = igual ou menor que 1 do total de linhas de código;
- WMC;
- DIT;
- NOC;
- RFC;

- CBO.

O objetivo deste trabalho é criar uma ferramenta que auxilie na auditoria de software entregue por empresas tercerizadas, contudo é percebe-se que ainda que o trabalho possua grande aprovação ele nunca substituira o fator humano da auditoria, o software serve apenas como uma ferramenta de avaliação geral do software entregue. Recomenda-se que uma vez que o software seja submetido à ferramenta e seja aceito é necessário que seja feita uma auditoria em cima de uma amostragem do software entregue sob o olhar de um analista especializado para tal atividade.
Uma vez analisado o projeto utilizando o Sonar Scanner um relatório é grerado e este relatório é exportado em formato XML para ser utilizado pela ferramenta desenvolvida neste trabalho. Uma segunda alternativa para capturar as informações geradas pelo Sonar seria utilizar o plugin Report do SonarQube para capturar a informação. Uma terceira alternativa seria criar um script python que leia a pagina html do sonarqube e através de expressão regular separar as informações da página.
As métricas "Violações do tipo Blocker", "Violações do tipo Major"e "Violações do tipo Minor"são estipuladas de acordo com o próprio perfil do Sonar, chamado Sonar Way. Contudo a melhor maneira seria criar um perfil com as regras da própria organização garantindo uma avaliação mais focada no objetivo do orgão.

4.2 Criação do *Dashboard*

Com as informações obtidas o dashboard funcionará como uma tela para visualização dessas métricas. A solução é composta por duas telas, uma mostrando uma visão geral do projetos, e com indicadores quanto aprovação ou reprovação de cada projeto seguindo os limites estabelecidos pelo edital [Mineral 2015] que é de onde está sendo baseado as métricas e limites dos indicadores. A segunda tela consiste em uma visão mais detalhada sobre cada projeto mostrando a evolução do projeto em cada métrica e com um *link* para o Sonar de cada métrica para um aprofundamento.

4.3 Avaliação

A avaliação do *dashboard* será feita por parte de um gestor de tecnologia de um orgão público. Ele avaliará aspectos de usabilidade da ferramenta e se a ferramenta possuiria condições mínimas de ser implantada. Caso não seja possível essa avaliação com um profissional da área a avaliação será feita através de professores que possuem tal experiência com contratação de software para orgãos públicos também avaliando aspectos como usabilidade e melhorias necessárias para implantação em um orgão. Para fazer está

avaliação o gestor responderá a um questionário contendo 15 perguntas referentes ao uso e funcionalidades do software produzido. O questionário é igual ao da imagem 10

4.4 Resumo do Capítulo

Inicialmente será criada uma instancia da ferramenta SonarQube que coletará as métricas nos projetos indicados. Quando essas métricas tiverem sido coletadas será gerado um arquivo XML contendo o nome do projeto, a métrica e o valor coletado. Esta primeira etapa é dada concluida quando XML for gerado. Para a segunda etapa é criado um *dashboard* onde será possível acompanhar dois projetos de software. O objetivo desta etapa é ter uma visão se um determinado projeto está dentro dos padrões estipulados ou não, para isto os projetos serão sinalizados quando estiverem em falta com alguma métrica. A última etapa deste trabalho consiste na avaliação do trabalho produzido que será feito juntamente com um gestor de projeto de um órgão público que responderá a um questionário quanto ao software.

Questionário Sobre o *Dashboard* de Monitoramento da Qualidade

Nome do Entrevistado: _____

Assinatura: _____

Data: ___ / ____/ _____

1) **Quanto tempo aproximadamente você demorou para identificar se um projeto estava em conformidade com os parâmetros estabelecidos ?**

_____ ( ) minutos ( ) segundos

2) Você conseguiria identificar claramente quais são as métricas utilizadas para a avaliação de um projeto ?

( ) SIM ( ) NÃO ( ) Algumas

3) Utilizando somente o *dashboard* você seria capaz de definir a métrica WMC ?

_____
_____
_____

4) Você conseguiria abrir o relatório do SonarQube de uma determinada métrica através do *dashboard*

( ) SIM ( ) NÃO

5) Na sua opinião o software apresentado possui aplicação no mundo real?

( ) SIM

( ) NÃO, porque _____
_____
_____

Figura 10 – Questionário a ser aplicado ao fim do período de teste

5 Metodologia

Este capítulo tem como objetivo apresentar a metodologia utilizada para desenvolvimento do TCC 1, e mostrar o plano base que servirá como guia para o desenvolvimento do TCC 2. A seção 5.1 apresenta a classificação da metodologia utilizada para o desenvolvimento do trabalho juntamente com o plano metodológico. A seção 5.2 descorre sobre o processo de desenvolvimento utilizado na criação da solução. E por último a seção 5.3 apresenta o cronograma utilizado no desenvolvimento do TCC 1 é um cronograma para o TCC 2.

5.1 Metodologia de Pesquisa

Segundo [Moresi e others 2003] a pesquisa pode ser entendida como um conjunto de ações que tem como objetivo encontrar um problema, construida através de procedimentos empíricos e sistemáticos. Para Moresi existem quatro classificações básicas para a pesquisa, quanto a sua natureza, sua abordagem, seu objetivo e o meio pelo qual é feita a investigação. A figura 11 apresenta em quais características este trabalho se apresenta.

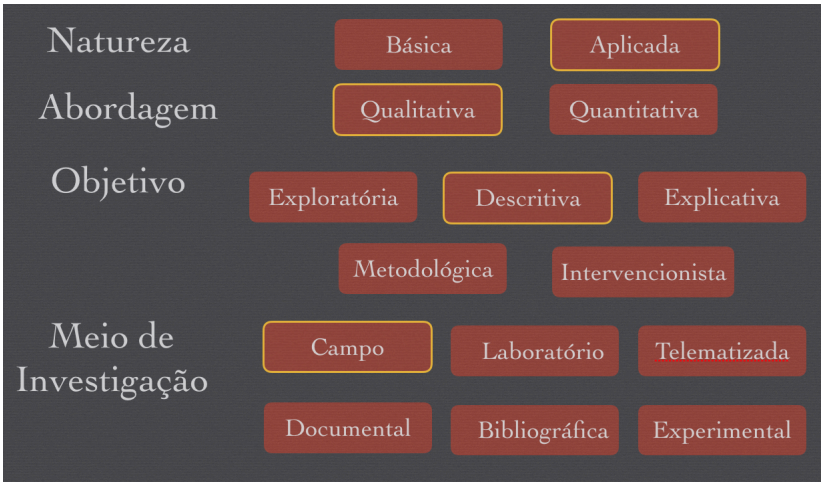

Figura 11 – Seleção das Características Metodológicas. Fonte: [Moresi e others 2003]

Este trabalho tem um caráter mais voltado para uma pesquisa aplicada por envolver características espécificas na contratação de software do governo brasileiro. Outra característica que determina este trabalho como uma pesquisa aplicada é a natureza do trabalho estar voltado para o uso nas áreas de TI dentro dos orgãos públicos.
Segundo [Gerhardt 2009] a pesquisa qualitativa é mais voltada para aspectos da realidade que não podem ser quantificados, mantendo o foco na compreensão. Neste aspecto o trabalho apresenta características qualitativas uma vez que a construção da solução é feita

de maneira dinâmica se adaptando as necessidades do usuário. Segundo as autoras outra característica inerente a este tipo de pesquisa é a observação do mundo social ao mundo natural, está característica se apresenta de maneira muito forte quando propor a solução foi adotado um conjunto de métricas que são utilizadas no mercado ao invés de outros conjuntos apresentados por outros autores.
Para Gil [Gil 2002] a pesquisa descritiva é focada em analisar características de uma população, fenômeno ou a relação entre as váriaveis as que compõem. Ests tipo de pesquisa não visa explicar os fenômenos que está descrevendo (apesar que a explicação para o fenômeno possa servir como base) [Moresi e others 2003]. Este trabalho apresenta o caráter descritivo ao se tratar da natureza das contratações de software para orgão brasileiros, ou quando se fala de um processo de manutenção de software.
Por último o meio de investigação a ser aplicado é a pesquisa de campo. Moresi [Moresi e others 2003] descreve a pesquisa de campo como sendo uma análise feita no local em que ocorre o fenômeno. Denise e Tatiana [Gerhardt 2009] acrescentam que a pesquisa de campo além de analisar o local do fenômeno também realiza coleta dos dados. Por se tratar da implantação da solução em um orgão todo o acompanhamento da implantação será feito com base nas características deste órgão e a coleta dos dados será referente à este órgão. O fato de que a solução será utilizada em um ambiente real garante que aplicação não será utilizada em um ambiente controlado descartando a pesquisa laboratorial.

5.1.1 Plano Metodológico

O plano metodológico consiste em duas fases: Iniciação e Execução. Durante o TCC1 será implementado somente a primeira fase e a fase de Execução será implementadaß no TCC2. A primeira fase (imagem 13) possui quatro atividades principais, são elas: Elaborar um roteiro de pesquisa, Pesquisar Referencias, Refinar Pesquisa e Catalogar material encontrado.
A atividade de Elaboar um Roteiro de Pesquisa consiste em encontrar a melhor string de busca. As strings foram montadas de acordo com o tema da pesquisa então não houve uma string geral que perpassa-se por todo o conhecimentodo trabalho. Essa escolha se deu pelo fato de que este trabalho passou por adaptações até que se chegasse ao trabalho que é hoje, então as primeiras strings eram feitas com conceitos chaves e já definidos sobre o que seria o trabalho, e com o decorrer o trabalho foi ganhando um escopo mais conciso.
A atividade de Elaborar Referencias e Refinar Pesquisa envolviam o processo de aplicar a string nas motores de busca selecionados, que no caso foram Google Scholar e Periódicos Capes. Uma vez aplicada a string o primeiro ponto a ser observado nos resultados era o título do material, caso o titúlo tivesse alguma relação com o tema pesquisado o artigo era separado (esse processo era válido somente para as duas primeiras páginas de resultados). Com os artigos separados lia-se os tópicos do artigo e havendo um conteúdo referente à pesquisa lia-se o artigo completo. Uma vez que era feita essa pesquisa gerava-se uma nova

string de busca com termos aproximados ou mais refinados e o processo se repetia. 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. Esse armazenamento é feito através de duas ferramentas de gerenciamento bibliográfico, o Zotero e o BibDesk. O Zotero foi utilizado para fazer a catalogação online dos materiais e gerar a bibliografia encontrada de cada material como mostra na figura 12.

- Área 1 - Categorização por pastas dos artigos encontrados.
- Área 2 - Artigos referentes à categoria seleciona. Dentro de cada artigo é possível encontrar a nota e um link para leitura do artigo selecionado.
- Área 3 - Informações do artigo selecionado.
- Área 4 - Tags referentes ao artigo.

Uma vez que o material era catalogado no Zotero ele era exportado para o Bibdeks por ter uma melhor integração com o Latex.

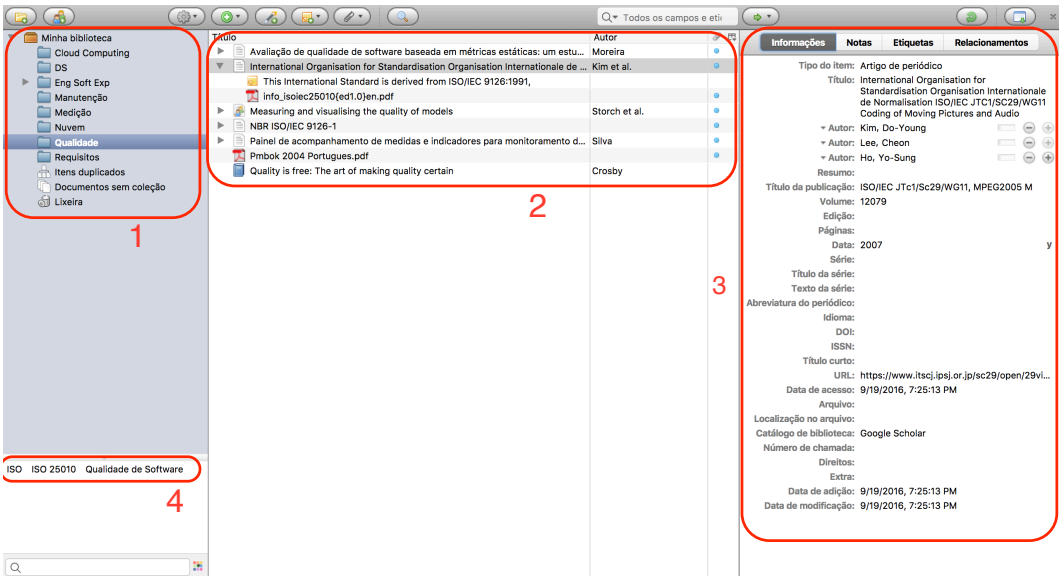

Figura 12 – *Screenshot* do Zotero contendo as categorias dos materiais pesquisados, suas *tags* e anotações

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

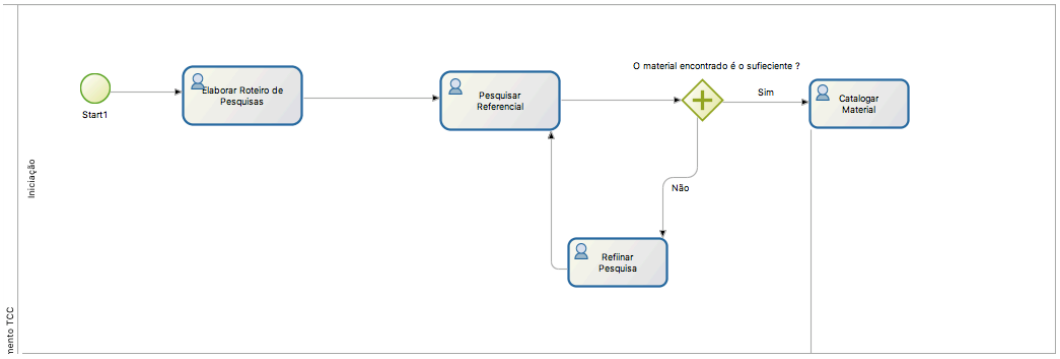

Figura 13 – Modelagem da Fase de Iniciação

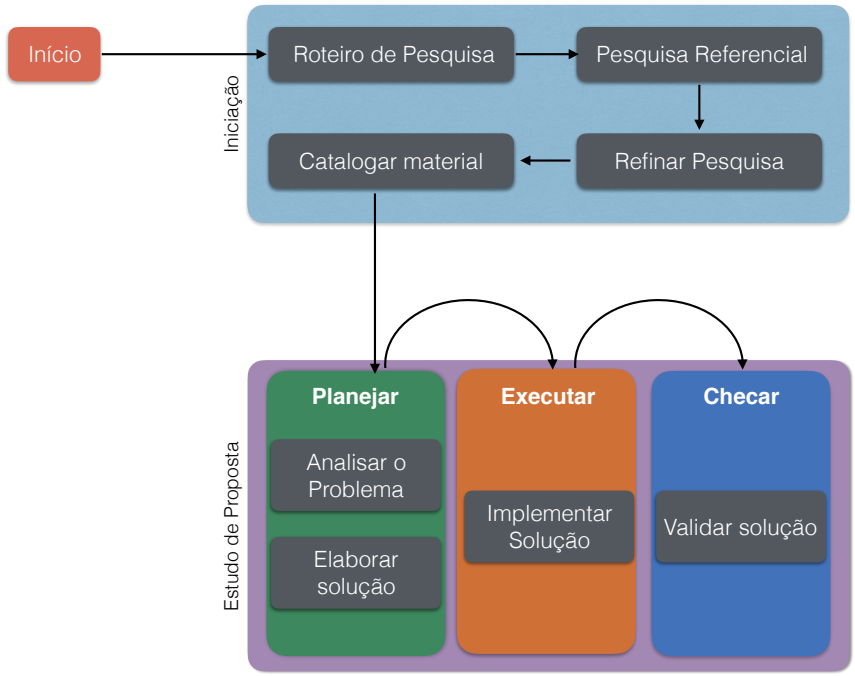

Figura 14 – Plano de Pesquisa

a solução final. No segundo momento implementa-se a solução e no terceiro momento acontece a validação dessa solução.

5.2 Metodologia de Desenvolvimento

O dashboard será criado utilizando de uma adaptação da metodologia de desenvolvimento Scrum. O Scrum é uma metodologia de desenvolvimento de software baseada em princípios de desenvolvimento ágil. As metodologias ágeis tem sido muito utilizadas em projetos com times pequenos, curto prazo de entrega do software e os requisitos são

constantemente alterados [Lopez-Martinez et al. 2016]. Como o Scrum é um modelo de desenvolvimento iterativo e incremental ele quebra o projeto de desenvolvimento em pequenas entregas chamadas de *sprints* que seriam pequenos ciclos de desenvolvimento que duram entre duas a quatro semanas. As *sprints* são consecutivas e nesse período algumas funcionalidades do sistema são implementadas e testadas [Pagotto et al. 2016]. Na figura 15 pode-se observar que o as funcionalidades desenvolvidas na *sprint* advêm de um escopo definido no início do projeto chamado de *product backlog* que é o conjunto de todas as funcionalidades do sistema, o conjunto de funcionalidades separadas para uma determinada *sprint* é chamada de *sprint backlog* [Sabbagh 2014].

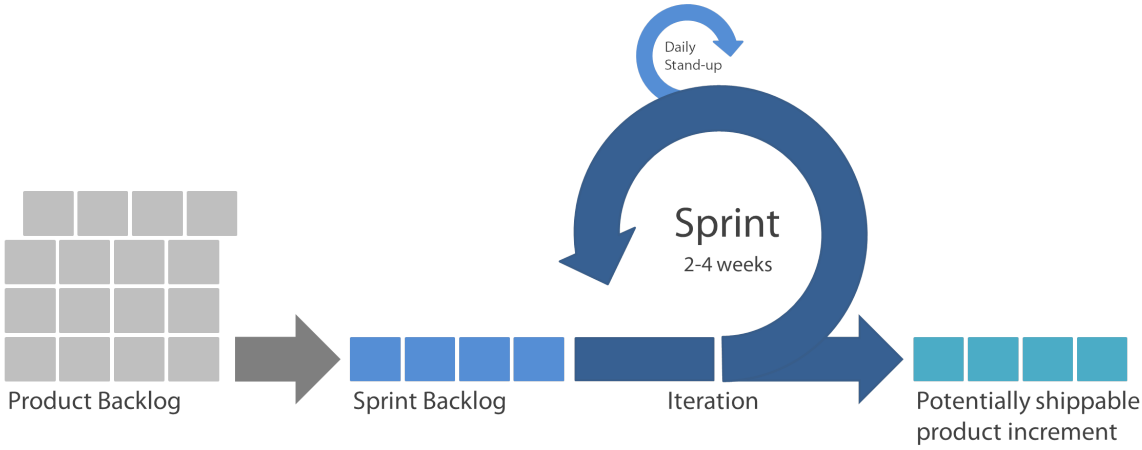

Figura 15 – Ciclo de Desenvolvimento do Scrum

Os principais conceitos que foram utilizados da metodologia foram:

- ***Product Backlog***: Lista de atividades que representam as funcionalidades que serão construidas no projeto. O *Product Owner* é quem escreve o *product backlog*, essas atividades são mutáveis ao decorrer do projeto, pois a equipe de desenvolvimento acaba conhecendo mais do produto [Sabbagh 2014].

- ***Sprint***: Ciclo de desenvolvimento com prazo definido em que são desenvolvidas as atividades do projeto. Neste trabalho definiu-se como sendo 15 dias o período referente a uma *Sprint*

- ***Sprint Backlog***: Atividades referentes à uma determinada *Sprint* na qual a equipe de desenvolvimento se compromete a entregar. Estas atividades são retiradas do *product backlog* [Mahnic 2011].

- **História do Usuário**: Descrição do ponto de vista do usuário sobre uma funcionalidade do produto. Este artefato é composto do que é conhecido como 3C's, Cartão, Conversa e Confirmação. O cartão é referente ao fato de se documentar a conversa em um cartão, este sendo acessível a todo o time de desenvolvimento. A conversa

é uma breve descrição da funcionalidade sob o olhar do usuário, um exemplo pode ser observado na figura 16. E a confirmação ou critérios de aceitação é um *checklist* abordando o que deve ser verificado para que a história seja dada como concluida.

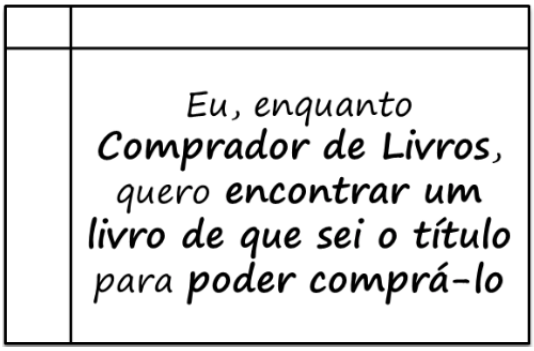

Figura 16 – Exemplo de História de Usuário. Fonte: [Sabbagh 2014]

- ***Story Point***:Unidade relativa que caracteriza o esforço da equipe de desenvolvimento para finalizar uma atividade. A escala utilizada é definida pela equipe de desenvolvimento, neste trabalho será utilizado a escala de Fibonacci (1,2,3,5,8,13, ...).
- ***Kanban***: Forma de visualização de atividades muito utilizadas com cartões que são movimentados em um quadro determinando o status da atividade como mostra a figura 17.

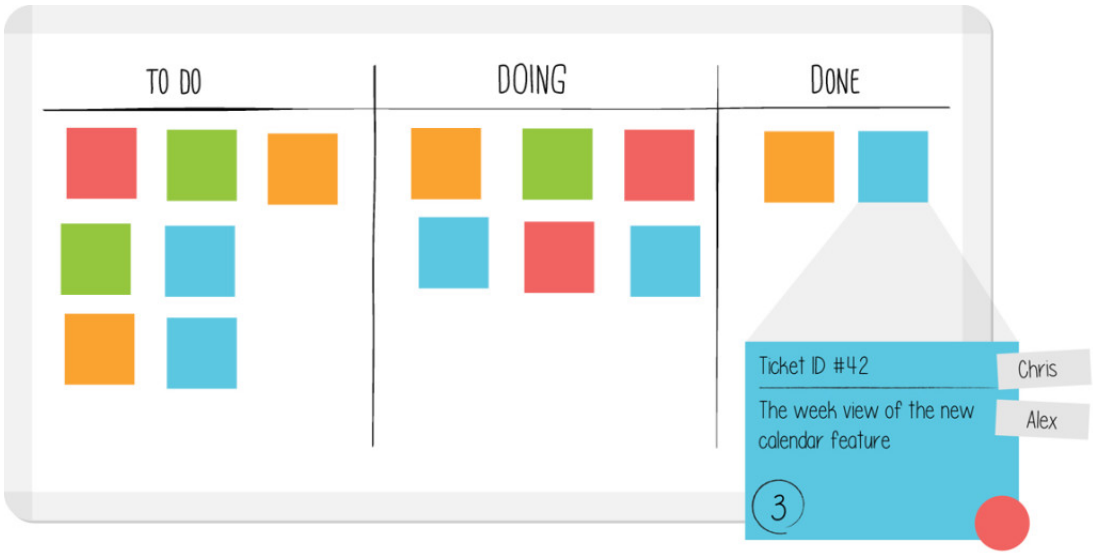

Figura 17 – Exemplo de *Kanban*

5.3 Cronograma

Tabela 3 – 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

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 39.

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 2 vezes nas páginas 21 y 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 31.

CHARTSJS documentation. <http://www.chartjs.org/docs/>. Accessed: 2016-10-10. Citado na página 42.

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 33.

FERENC, R. et al. Source Meter Sonar Qube Plug-in. In: . [S.l.]: IEEE, 2014. p. 77–82. Citado na página 40.

FEW, S. *Information Dashboard Design - The effective Visual Comunication of Data*. [S.l.]: O'Reilly, 2006. Citado na página 38.

FILHO, C. F. *História da computação: O Caminho do Pensamento e da Tecnologia*. [S.l.]: EDIPUCRS, 2007. Citado na página 21.

GERHARDT, D. T. S. T. E. *Métodos de pesquisa*. [S.l.]: Universidade Aberta do Brasil, 2009. Citado 2 vezes nas páginas 49 y 50.

GIL, A. C. *Como elaborar projetos de pesquisa*. São Paulo (SP): Atlas, 2002. OCLC: 817765297. ISBN 978-85-224-3169-4. Citado na página 50.

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 2 vezes nas páginas 13 y 36.

GOOGLE Charts documentation. <https://developers.google.com/chart/interactive/docs/reference>. Accessed: 2016-10-10. Citado na página 42.

GRAČANIN, D.; MATKOVIĆ, K.; ELTOWEISSY, M. Software visualization. *Innovations in Systems and Software Engineering*, v. 1, n. 2, p. 221–230, set. 2005. ISSN 1614-5046, 1614-5054. Disponível em: <http://link.springer.com/10.1007/s11334-005-0019-8>. Citado 2 vezes nas páginas 36 y 37.

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. ISSN 0941-0643, 1433-3058. Disponível em: <http://link.springer.com/10.1007/s00521-013-1481-0>. Citado na página 36.

ISO 25010. *Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models.* 2011. ed. [S.l.]. Citado 3 vezes nas páginas 13, 30 y 31.

KANER, C. Software engineering metrics: What do they measure and how do we know? *10TH INTERNATIONAL SOFTWARE METRICS SYMPOSIUM*, 2004. Citado na página 33.

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 32.

LOPEZ-MARTINEZ, J. et al. Problems in the Adoption of Agile-Scrum Methodologies: A Systematic Literature Review. In: . [S.l.]: IEEE, 2016. p. 141–148. Citado na página 53.

MACHADO, M. P.; SOUZA, S. F. Métricas e qualidade de software. *Departamento de Informática–Universidade Federal do Espírito Santo*, 2004. Citado na página 21.

MAHNIC, V. A case study on agile estimating and planning using scrum. *Elektronika ir Elektrotechnika*, v. 111, n. 5, p. 123–128, 2011. Disponível em: <http://ecoman.ktu.lt/index.php/elt/article/view/372>. Citado na página 53.

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 39.

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 33 y 34.

MINERAL, D. N. de P. *Edital Pregão Eletronico No 14/2015*. [S.l.], 2015. Citado 4 vezes nas páginas 15, 27, 45 y 46.

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 34.

MORESI, E.; others. Metodologia da pesquisa. *Brasília: Universidade Católica de Brasília*, v. 108, 2003. Disponível em: <http://ftp.unisc.br/portal/upload/com_arquivo/1370886616.pdf>. Citado 3 vezes nas páginas 13, 49 y 50.

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, 28 y 29.

PADUELLI, M. M. *Manutenção de Software: problemas típicos e diretrizes para uma disciplina específica*. Dissertação (Mestrado), 2007. Citado 2 vezes nas páginas 21 y 28.

PAGOTTO, T. et al. Scrum solo: Software process for individual development. In: *Information Systems and Technologies (CISTI), 2016 11th Iberian Conference on*. [S.l.]: AISTI, 2016. p. 1–6. Citado na página 53.

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 32.

PIGOSKI, T. M. Pratical software maintenance: Best practices for managing your software investment. *Wiley Computer Publishing*, 1996. Citado na página 33.

PRESSMAN, R. S. *Software Engineering - A Practioner's Approach*. 7. ed. [S.l.]: Higher Education, 2010. Citado 2 vezes nas páginas 31 y 33.

SABBAGH, R. *Scrum: Gestão ágil para projetos de sucesso*. Editora Casa do Código, 2014. Disponível em: <https://books.google.com/books?hl=en&lr=&id=pG-CCwAAQBAJ&oi=fnd&pg=PT9&dq=%22Agilidade+em+diferentes+organiza%C3%A7%C3%B5es.+Hoje+%C3%A9+poss%C3%ADvel+ver+os+assuntos+ligados%22+%22Mas,+segundo+o+pr%C3%B3prio+Rafael%22+%22que+n%C3%A3o+%C3%A9+necess%C3%A1rio+ter+uma+ampla+compreens%C3%A3o+do+Scrum+para%22+&ots=ERSywNIEAd&sig=rvmJD_BXcQY-P0MYZBqQJOaOcOA>. Citado 3 vezes nas páginas 13, 53 y 54.

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*. IEEE, 2016. p. 1–6. Disponível em: <http://ieeexplore.ieee.org/abstract/document/7549475/>. Citado na página 37.

SCHAIDT, Y. R. L. *Monitoramento de Qualidade de Software na Administração Pública Federal*. Dissertação (Mestrado) — Universidade de Brasília, 2014. Citado 2 vezes nas páginas 22 y 29.

SILVA, A. B. Painel de acompanhamento de medidas e indicadores para monitoramento da qualidade de software. 2014. Disponível em: <http://bdm.unb.br/handle/10483/6979>. Citado na página 22.

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. Disponível em: <http://reuse.cos.ufrj.br/files/publicacoes/graduacao/PF_Marlon_IAVEMS.pdf>. Citado na página 37.

SOMMERVILE, I. *Software Engineering*. 9. ed. [S.l.]: Pearson, 2011. Citado na página 31.

SONARQUBE documentation. <http://docs.sonarqube.org/display/SONAR/Architecture+and+Integration>. Accessed: 2016-10-10. Citado 3 vezes nas páginas 13, 40 y 41.

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