

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-

68 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, 15 de novembro de 2016:

Prof. Dr. Maurício Serrano
Orientador

Profa. Dra. Milene Serrano
Coorientadora

Ajax
Convidado 1

André Barros
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.

Listas de ilustrações

Figura 1 – Relação entre as NBR ISO/IEC 9126 e NBR ISO/IEC 14598 .Fonte: [NBR ISO/IEC 9126-1 2016]	28
Figura 2 – Modelo de Qualidade para Qualidade Interna e Externa .Fonte: [NBR ISO/IEC 9126-1 2016]	29
Figura 3 – Produto de Qualidade de Software.Fonte: [ISO 25010]	31
Figura 4 – Diagrama das Atividades de Manutenção de Software.Fonte: [Pfleeger e Bohner 1990]	33
Figura 5 – Exemplo de glifo utilizado na visualização da variação das medidas relacionadas com o desenvolvimento computacional. Fonte: [Gomes e Tavares 2011]	37
Figura 6 – Comparativo entre a)Bifocal Display e b) Perspective Wall	37
Figura 7 – Tipos de Visualização Mais Frequentes. Fonte: [Schwendimann et al. 2016]	38
Figura 8 – Exemplo de Gráfico de Barra Utilizando Google Charts	39
Figura 9 – Exemplo de Gráfico de Linhas utilizando HiCharts	39
Figura 10 – Exemplo de Gráfico de Pizza utilizando Chart.js	40
Figura 11 – Exemplo de <i>Network Graph</i> utilizando Google Charts	40
Figura 12 – Exemplo 1 de <i>Dashboard</i> apresentado por Stephen Few. Fonte: [Few 2006]	41
Figura 13 – Exemplo 2 de <i>Dashboard</i> apresentado por Stephen Few. Fonte: [Few 2006]	42
Figura 14 – Exemplo 3 de <i>Dashboard</i> apresentado por Stephen Few. Fonte: [Few 2006]	42
Figura 15 – Sete aspectos de qualidade de código cobertos pelo SonarQube. Fonte: [SonarQube documentation]	46
Figura 16 – Arquitetura SonarQube composta de quatro componentes: [SonarQube documentation]	47
Figura 17 – Integração SonarQube em Diversas Áreas de Desenvolvimento de Software: [SonarQube documentation]	47
Figura 18 – Questionário a ser aplicado ao fim do período de teste	53
Figura 19 – Seleção das Características Metodológicas. Fonte: [Moresi e others 2003]	55
Figura 20 – Modelagem da Fase de Iniciação	56
Figura 21 – <i>Screenshot</i> do Zotero contendo as categorias dos materiais pesquisados, suas <i>tags</i> e anotações	58
Figura 22 – Modelagem da Fase de Execução	58
Figura 23 – Plano de Pesquisa	59
Figura 24 – Ciclo de Desenvolvimento do Scrum	60

Figura 25 – Exemplo de História de Usuário. Fonte: [Sabbagh 2014]	61
Figura 26 – Exemplo de <i>Kanban</i>	61

Lista de tabelas

Tabela 1 – Índice de Cobertura Por Tipo de Teste do Edital da DNPM. Fonte: [Mineral 2015]	27
Tabela 2 – Métricas de Qualidade de Código Exigidas pela DNPM. Fonte: [Mineral 2015]	27
Tabela 3 – Mapeamento de Cores e seus Significados	41
Tabela 4 – Cronograma TCC1	62
Tabela 5 – Cronograma TCC2	62
Tabela 6 – <i>Status</i> Completude do Trabalho	64

Listas 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	<i>Weighted Methods per Class</i>
DIT	<i>Depth of Inheritance</i>
NOC	<i>Number of Children</i>
CBO	<i>Coupling Between Objects</i>
RFC	<i>Response for a class</i>
LCOM	<i>Lack of Cohesion in Methods</i>
DNPM	Departamento Nacional de Produção Mineral
UI	<i>User Interface</i>
SCM	<i>Software Configuration Management</i>
API	<i>Application Programming Interface</i>
TCC	Trabalho de Conclusão de Curso

Sumário

1	INTRODUÇÃO	21
1.1	Contextualização	21
1.2	Problema de pesquisa	22
1.3	Justificativa	22
1.4	Objetivos	23
1.4.1	Objetivos Gerais	23
1.4.2	Objetivos Específicos	23
1.5	Resultados Esperados	24
1.6	Organização do trabalho	24
2	REFERENCIAL TEÓRICO	25
2.1	Processo de Contratação de Software na APF	25
2.1.1	Avaliação da Qualidade Em Um Processo de Contratação	27
2.2	Qualidade	28
2.2.1	Norma SQuaRE	29
2.2.2	Manutenção de Software	31
2.3	Métricas de Qualidade de Software	33
2.3.1	Suíte de Chidamber-Kemerer	33
2.3.2	Suíte MOOD	35
2.4	Visualização da Informação	36
2.4.1	Visualização de Dados	36
2.4.2	Dashboard	38
2.5	Resumo do Capítulo	43
3	SUPORTE TECNOLÓGICO	45
3.1	Ferramentas de Programação	45
3.1.1	GIT	45
3.1.2	Github	45
3.1.3	SonarQube	46
3.1.4	Google Charts e Charts.js	48
3.2	Ferramentas de Gerenciamento	48
3.2.1	Bonita	48
3.2.2	Mac OS X	49
3.2.3	LaTeX	49
3.2.4	Sublime Text 3	49
3.2.5	Zotero	49

3.3	Resumo do Capítulo	49
4	PROPOSTA	51
4.1	Coleta das Métricas	51
4.2	Criação do <i>Dashboard</i>	52
4.3	Avaliação	52
4.4	Resumo do Capítulo	52
5	METODOLOGIA	55
5.1	Metodologia de Pesquisa	55
5.1.1	Plano Metodológico	56
5.2	Metodologia de Desenvolvimento	59
5.3	Cronograma	61
5.3.1	Cronograma TCC1	62
5.3.2	Cronograma TCC2	62
5.4	Resumo do Capítulo	62
6	RESULTADOS PARCIAIS	63
6.0.1	<i>Status</i> atual do Projeto	63
6.0.2	Próximas Atividades	63
	REFERÊNCIAS	65

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 problemática (Problema) . As justificativas que levaram à realização deste trabalho são apresentadas na seção Justificativa deste mesmo capítulo. Após as justificativas, são apresentados os objetivos (Objetivos) que servem como guia para solução proposta. Resultados esperados (Resultados Esperados) Os resultados dessa pesquisa são apresentados na seção tal, Resultados Esperados. Por fim, tem-se a Organização do Trabalho, a qual procura apresentar os principais capítulos dessa monografia.

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 usuários, fabricantes e pesquisadores que debatiam sobre os constantes problemas no desenvolvimento de software [Paduelli 2007]. Desde então, empresas, órgãos públicos e diversas outras instituições utilizam o computador para automatizar tarefas ou cálculos antes feitos por humanos [Filho 2007]. O Decreto nº.271 de 1997 [BRASIL 1997] coloca a atividade de informática (e seus afins) como sendo um dos tipos de serviço passíveis de terceirização, ou seja, uma empresa terceirizada deve realizar os serviços referentes a este tipo de atividade. Uma vez que o software é produzido por uma empresa terceirizada 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, tendo como os principais pilares três áreas comuns da Engenharia de Software, Gerência de Configuração, Integração Contínua e Análise Estática de Código. Uma solução próxima a essa vem sendo trabalhada dentro de alguns Órgãos Públicos do Governo Federal, entre eles o Tribunal de Contas da União. Este órgão tem mostrado bons resultados nesta área. O problema encontrado atualmente está na falta de um acompanhamento na qualidade dos softwares entregues pelas empresas terceirizadas

1.2 Problema de pesquisa

O principal produto da Engenharia de Software é o software. Contudo o que se tem vivenciado na realidade brasileira de computação é que o software que está sendo entregue é um software precário e de baixa qualidade.

Por ser um conceito abstrato, qualidade é entendida como algo amplo. Porém, está, normalmente, associada a uma medida relativa. Nesse sentido, qualidade por ser entendida como "conformidade às especificações" [ref]. Uma vez conceituado dessa forma, a não conformidade às especificações é vista como baixa ou ausência de qualidade.

Uma das grandes dificuldades nos Órgãos Públícos está no acompanhamento das manutenções prestadas por terceirizadas. Esse problema se agrava ainda mais quando a empresa contratante não consegue acompanhar ou não tem parâmetros concretos de indicadores de qualidade. Este trabalho tem como proposta a criação de um dashboard de monitoramento para softwares legados. Com esse dashboard, pretende-se que seja possível acompanhar mais facilmente, através de recursos visuais, indicadores de qualidade de código nos projetos monitorados.

O Dentre os desafios dessa proposta, tem-se a necessidade de prover uma visualização de dados de forma simplificada e objetiva, visando o acompanhamento dos indicadores de qualidade em um Órgão Público Federal. Lembrando que, normalmente, os Órgãos Públícos Brasileiros não são os desenvolvedores diretos dos produtos de software. Tais demandas são terceirizadas para empresas especializadas, o que enfatiza a necessidade de se ter um suporte que facilite a avaliação dos entregáveis por parte dos Órgãos Públícos. Tomando este problema como base, tem-se a seguinte questão de pesquisa:

"Definido um conjunto de métricas, como criar um dashboard que avalie a qualidade de software de um órgão público federal?"

1.3 Justificativa

continuaram. Essas lacunas estão, principalmente, associadas à apresentação A motivação deste trabalho deu-se como uma extensão de um trabalho de conclusão de concurso elaborado anteriormente por Luiza e Yago [Schaidt 2014]. Nesse trabalho, eram tratados aspectos de monitoramento da qualidade, aspectos de contratação de software dentro desses órgãos e como acontecia o acompanhamento desse software. O trabalho também propunha uma solução com integração contínua e gerência de configuração, dentro de um Órgão Público Federal. Após a conclusão do trabalho, algumas lacunas continuaram. Essas lacunas estão, principalmente, associadas à apresentação destes dados para a equipe de gestão, com o intuito de facilitar o acompanhamento das métricas.

Outro trabalho também conferiu base para a proposta. Trata-se do trabalho de conclusão de curso de Adriano [Silva 2014]. Em seu trabalho Adriano apresenta a construção de um painel para acompanhamento de métricas, voltadas para qualidade de software na área de teste de software que seriam implementadas para o laboratório CQTS. Contudo, a presente proposta difere de [Silva 2014], uma vez que são focos: facilitar a comunicação entre as terceirizadas e os gestores de projeto, provendo um suporte que possibilitará a intermediação entre essas partes; e permitir que a qualidade seja aferida pelos gestores de projeto - membros dos Órgãos Públicos - através de um suporte com recursos visuais e orientado por indicadores de qualidade.

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 terceirizada e o gestor de projetos apresentando uma ferramenta que fará a ponte entre as duas partes.

1.4 Objetivos

1.4.1 Objetivos Gerais

O objetivo deste trabalho é propor um *dashboard*, que com auxílio de recursos visuais auxilie no processo de contratação de software. O público alvo desta solução são gestores de projeto, e líderes de setores das áreas de tecnologia dos Órgãos Públicos Federais Brasileiros. A solução engloba a Visualização da Informação através de um *dashboard* em que são mostrados métricas e indicadores previamente definidos pelo gestor. Estas métricas são escolhidas pelo gestor de um conjunto de métricas que são apresentadas pela solução. Espera-se que com o *dashboard*, os gestores dos projetos tenham mais visibilidade do processo de construção ou manutenção do software prestado pelas terceirizadas.

1.4.2 Objetivos Específicos

Para que seja possível alcançar o objetivo geral, alguns objetivos mais específicos precisam ser levados em consideração. São eles:

- Definir um conjunto de métricas que possam ser utilizadas pelo gestor como indicadores da qualidade de código do software em análise.
- Utilizar dados de uma ferramenta de análise estática para coleta de métricas.
- Propor um dashboard de visualização e acompanhamento de qualidade de código, instanciando-o para um projeto específico. Nesse caso, será necessário simular um ambiente, configurado com base em um Órgão Público Federal.

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 análise estática. Essa visualização ocorrerá através de um *dashboard* que indica a atual situação de um projeto tendo como indicadores métricas estabelecidas por um gestor de tecnologia.

1.6 Organização do trabalho

A monografia está organizada em capítulos. No primeiro capítulo, é apresentada uma **Introdução** ao trabalho, destacando a problemática e os objetivos a serem alcançados. No segundo capítulo, encontra-se o **Referencial Teórico** que é necessário para que se possa entender o motivo das decisões tomadas neste trabalho. O terceiro capítulo apresenta uma descrição das **Ferramentas** que foram utilizadas neste trabalho, destacando, principalmente o SonarQube. No quarto capítulo, é apresentada a **Proposta** deste trabalho, é neste capítulo que é detalhado um pouco mais a pesquisa em andamento. O quinto capítulo é voltado para a **Metodologia** a qual foi e será aplicada para que se possa alcançar os objetivos estabelecidos. No sexto e último capítulo, são apresentados os status das atividades concluídas e/ou em andamento bem como os principais resultados obtidos até o momento.

2 Referencial Teórico

Este capítulo tem como principal objetivo conferir detalhes sobre o referencial teórico, o qual embasa a presente proposta. Inicialmente, o capítulo apresenta uma visão geral sobre o processo de contratação de software na Administração Pública Federal (APF), onde são destacadas algumas leis que fundamentam o tópico de contratação no âmbito das terceirizadas bem como estabelecem o nível de qualidade exigido por parte dos Órgãos Públicos Brasileiros. Em seguida discute-se sobre o conceito de qualidade de software na visão de alguns atores com destaque para o que é qualidade de acordo com a ISO 9126. Essa serviu como base para a norma SQUARE também discutida neste trabalho. Após a definição de qualidade pela norma SQUARE, sendo apresentados conceitos fundamentais de manutenção de software, os quais servem como guias para o desenvolvimento desta proposta. Tendo definido manutenibilidade, alguns conjuntos de métricas, que visam identificar possíveis lacunas de manutenibilidade 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 nº 2.711 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 terceirizados. O Decreto coloca como exemplo algumas atividades, tais como: conservação, limpeza, segurança, vigilância, transporte, informática, copeiragem, recepção, reprografia, telecomunicações. Essas atividades são 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 nº 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 acordo com 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 Órgão contratante. Para ajudar no processo de contratação de empresas terceirizadas voltadas para a área de Tecnologia da Informação (TI), o Tribunal de Contas da União 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 produtos de software, 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 à 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 terceirizada, o mesmo passa por um processo de verificação de todos os artefatos que foram entregues, sejam eles em forma de documento, ou em código. São lançados dezenas de editais todos os anos com o intuito de suprir a necessidade dos Órgãos Públicos quanto à 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 terceirizada. Em um edital feito pelo Departamento Nacional de Produção Mineral [Mineral 2015], um dos objetos de avaliação é a cobertura mínima 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, Tabela 2. Nessa, são mostrados outros detalhes que 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), as quais não possuem valores definidos de cobrança, sendo, portanto, ajustados conforme 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 entregue é precário e de baixa qualidade. Por ser um conceito abstrato, qualidade é algo amplo. Porém, normalmente, está associada a uma medida relativa, sendo entendida como "conformidade às especificações". Conceituando dessa forma, a não conformidade às especificação pode ser vista como baixa ou 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, conforme ilustra a Figura 1. A primeira categoria está relacionada a qualidade interna e a qualidade externa do software. A segunda categoria relaciona-se com a qualidade de uso do software [NBR ISO/IEC 9126-1 2016]

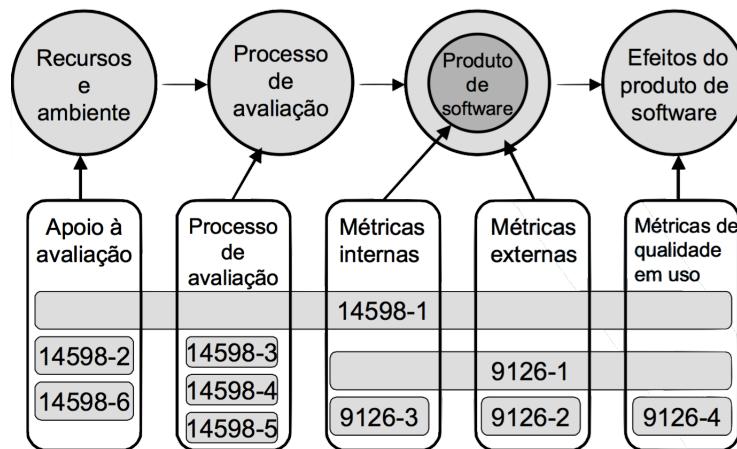


Figura 1 – Relação entre as NBR ISO/IEC 9126 e NBR ISO/IEC 14598 .Fonte: [NBR ISO/IEC 9126-1 2016]

Como 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 análise estática do código fonte, revisão dos documentos produzidos, entre outros. A qualidade externa, por sua vez, já apresenta foco mais voltado para as relações externas do software, normalmente, relacionando-se com a execução do código. Nesse caso, é necessário coletar 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, medidas por meio de métricas internas e externas.

Segundo a ISO 9126, essas características podem ser definidas como:

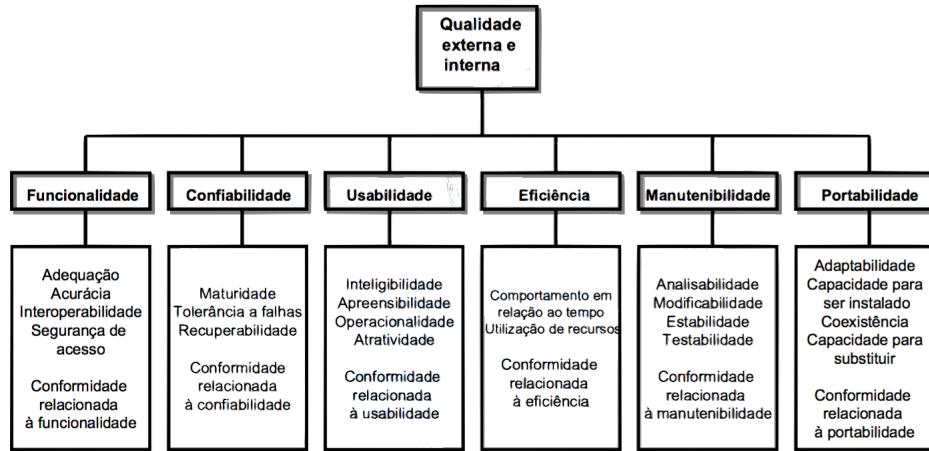


Figura 2 – Modelo de Qualidade para Qualidade Interna e Externa .Fonte: [NBR ISO/IEC 9126-1 2016]

- **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 conhecido como SQuaRE. Esse conjunto trazia um *framework* aprimorado à atual norma vigente. Além disso, 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 sob o 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, à Qualidade em Uso, e o último, à Qualidade de Dados. O modelo de Qualidade do Produto subdivide um sistema de software em oito categorias, como mostra a Figura 3. Assim como a ISO 9126, a ISO 25010 também apresenta categorias, estas categorias assemelham-se às categorias da ISO 9126, sendo essa base para criação da norma SQuaRE. Seguem as características:

- **Adequação Funcional:** nível que determina o quanto um produto ou sistema satisfazem as especificações providas pelo usuário.
- **Eficiência de Desempenho:** desempenho relativo à quantidade de recursos usados em condições específicas.
- **Compatibilidade:** o nível que um sistema ou produto pode compartilhar informações, 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 específico.
- **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.

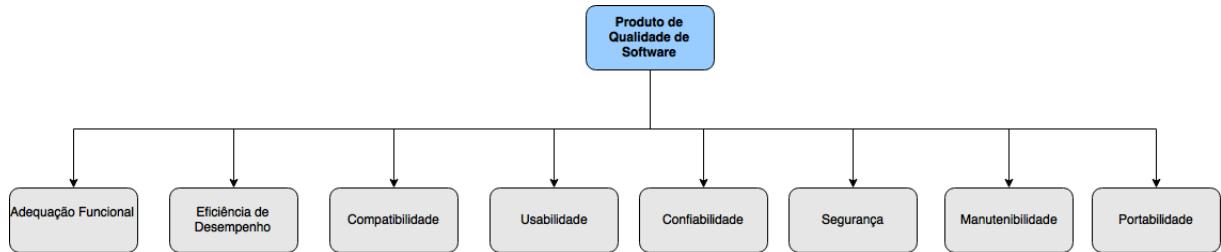


Figura 3 – Produto de Qualidade de Software. Fonte: [ISO 25010]

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

2.2.2 Manutenção de Software

Segundo Sommerville [Sommerville 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, até mudanças significativamente grandes para correção 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, e adaptação do software para lidar com uma mudança do ambiente e aperfeiçoar as funcionalidades em atendimento 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 Sommerville, 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:** feita 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 desempenho 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 e Bohner [Pfleeger e Bohner 1990] 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:

- **Análise do Impacto da Mudança de Software:** estima o impacto de uma determinada mudança. Nesta atividade, determina-se o grau de mudança e o quanto esta 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 rápido se dá o processo de análise do software.
- **Implementação da Mudança:** incremento ou modificação do software. Esta 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 manutenibilidade de software apresentada pela norma SQuaRE.
- **Mudanças pelo Efeito Cascata:** Análise da propagação das mudanças ao longo do software. Essa atividade está intimamente relacionada ao indicador de coesão, que relaciona a responsabilidade de uma classe com seus métodos, e acoplamento do software, este afere o quanto 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 Kusters e Heemstra [Kusters e Heemstra 2001] mostra as dificuldades atuais na manutenção de software com base em seis grandes organizações da Alemanha. Um dos resultados obtidos foi que existe uma falta muito grande na percepção quanto ao tamanho e ao custo das manutenções de software. Os autores relatam

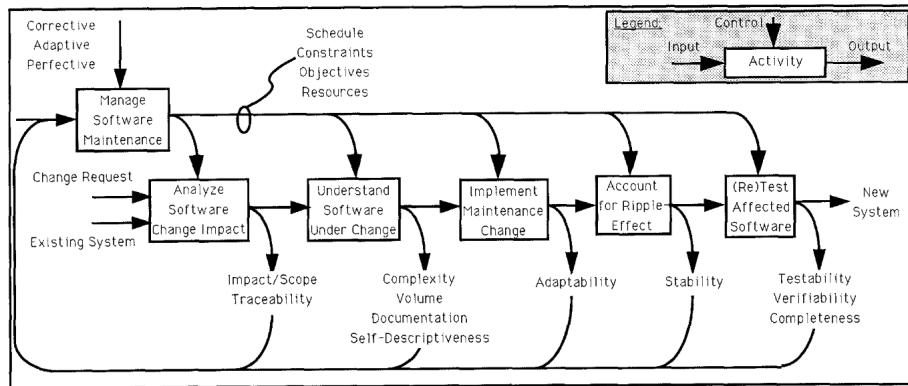


Figura 4 – Diagrama das Atividades de Manutenção de Software. Fonte: [Pfleeger e Bohner 1990]

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. Métrica de qualidade de software é uma função; cuja entrada é uma informação de software, e cuja saída é um único valor numérico que pode ser interpretado como o nível que é dado a um software [Kaner 2004]. Segundo [Pressman 2010], métricas podem ser definidas como sendo um pequeno subconjunto de informações úteis acerca do software.

Segundo Mills, algumas características são inerentes a uma boa métrica, tais 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.

$$WMC = \sum_{i=1}^n Ci \quad (2.1)$$

Essa métrica serve como indicador para o nível geral da modularização. Quanto maior o valor, mais complexa está a classe ou poucas classes a classe ou poucas classes possuem um índice 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 análise, seja por herança ou implementação. Essa métrica junto com outras ajuda a determinar quais classes são primordiais para o funcionamento do sistema.
- **CBO:** Esta métrica é dada pela equação 2.2:

$$CBO = \frac{NumberOfDependencies}{NumberOfClassInPackage} \quad (2.2)$$

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. Esta medida é muito utilizada para cobertura de testes, em que pode ser constatada 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:

$$P = \{(Ii, Ij) | (Ii \cap Ij) = \emptyset\} \quad (2.3)$$

e Q o conjunto tal que:

$$Q = \{(Ii, Ij) | (Ii \cap Ij) = \emptyset\} \quad (2.4)$$

então, LCOM é definida como sendo:

$$LCOM = |P| - |Q| se |P| > |Q| \quad (2.5)$$

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. Dessa forma, princípios como polimorfismo, baixo acoplamento, encapsulamento e outros são altamente valorizados. Com base em Meirelles [Meirelles 2008] e Moreira [Moreira 2015], as métricas são:

- **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

$$Mv(C) \quad (2.6)$$

Número de métodos encapsulados em uma classe C

$$Me(C) \quad (2.7)$$

O total de métodos é dado por

$$Mt(C) = Mv(C) + Me(C) \quad (2.8)$$

A equação que representa esta métrica é dada por:

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

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

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

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

$$MIF = \frac{TMh}{TMD} \quad (2.11)$$

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

$$MIF = \frac{TAh}{TAd} \quad (2.12)$$

- **CFA:** Razão entre o total de acoplamentos permitidos no sistema e o atual número de acoplamentos possíveis por herança. Para esta métrica, 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.

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

- **PFA:** Razão entre o número 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 usuário. 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 tornaram-se peças fundamentais do 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 esta 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, por sua vez, representa uma entidade ou um elemento do todo. O *glifo* tem uma forma que é definida pelo valor dos atributos da entidade conforme ilustrado na Figura 5.

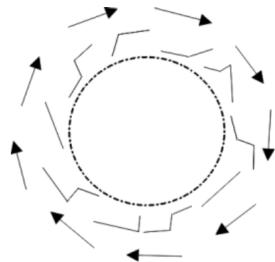


Figura 5 – Exemplo de glifo utilizado na visualização da variação das medidas relacionadas com o desenvolvimento computacional. Fonte: [Gomes e Tavares 2011]

- **Visualização de Estruturas e Relações:** esta técnica utiliza-se do conjunto de dados ou elementos que podem ser distribuídos de maneira hierárquica. Dentro deste conceito, existe uma ramificação para o uso da técnica, aplicada com 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 contém 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 Figura 6.

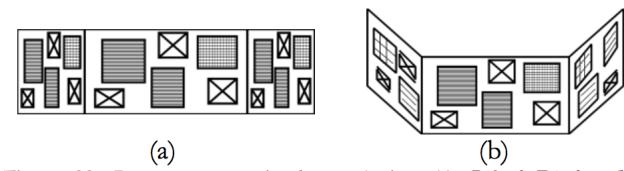


Figura 6 – Comparativo entre a) Bifocal Display e b) Perspective Wall

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. A presente proposta tem seu objetivo focado na segunda categoria, ou seja, em 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], os autores apontam 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 decompõe em módulos, e cada módulo responsável por uma parte do código.
- **Atributos:** Deve conter o número da versão, tamanho, alterações, complexidade.

Uma vez que são coletados os dados, esses já estão categorizados de acordo com a sua natureza, e podem ser exibidos. Mas, 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. Deve ser construído 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.

Schwendimann [Schwendimann et al. 2016] apresenta uma revisão sistemática sobre *dashboard* para aprendizado. A pesquisa foi feita em 55 artigos. Na Figura 7 pode-se ver que os três tipos de visualização mais utilizados são gráfico de barras (33 artigos), gráfico de linhas (24 artigos) e tabelas (21 artigos).

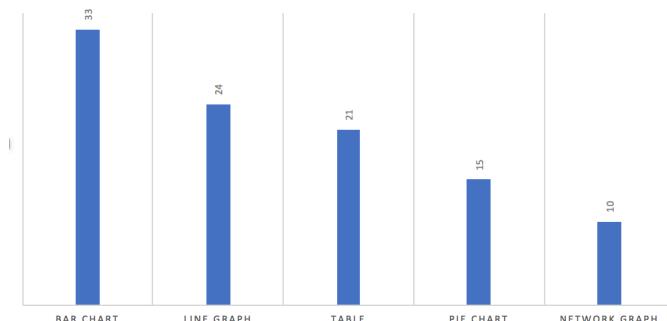


Figura 7 – Tipos de Visualização Mais Frequente. Fonte: [Schwendimann et al. 2016]

- Gráfico de Barras: representação de quantidade ao longo de uma escala numérica (Figura 8). Para que haja uma comparação significativa, deve ser feita sob a perspectiva de uma escala linear, partindo de zero [Doumont e Vandebroeck 2002]

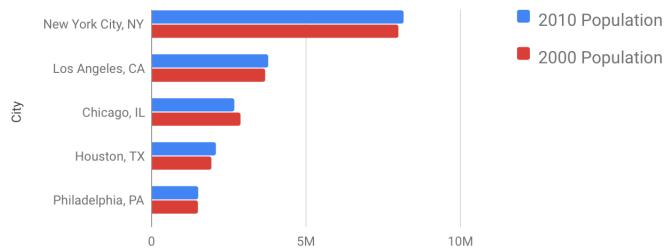


Figura 8 – Exemplo de Gráfico de Barra Utilizando Google Charts

- Gráfico de Linhas: muito utilizado quando se tem um conjunto de dados contínuos. Pode ainda ser utilizado para determinar padrões ou uma tendência (Figura 9). Normalmente, representam dados relacionados à tempo.

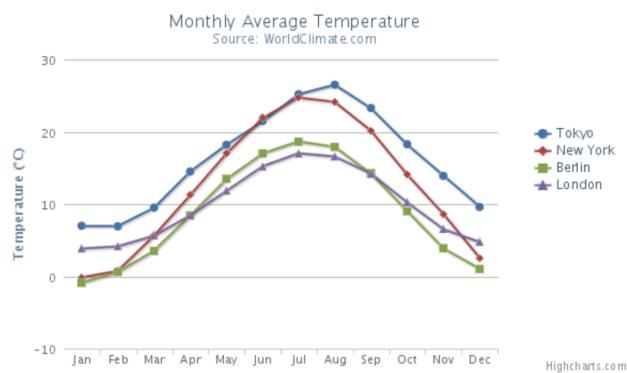


Figura 9 – Exemplo de Gráfico de Linhas utilizando HiCharts

- Gráfico de Pizza: é melhor usado quando se deseja comparar um setor em relação ao total. O gráfico de pizza é um gráfico circular dividido em segmentos. Cada segmento representa uma categoria e o somatório dessas partes formam o todo (Figura 10). Não se aconselha utilizar o gráfico de pizza quando se tem que representar mais de sete categorias.
- *Network Chart*: este tipo de visualização reforça relacionamentos entre entidades. As entidades são mostradas como nós e os relacionamentos como linhas (Figura 11).

Outro ponto importante na criação do *dashboard* é o uso da paleta de cores. Ying Li e Anshul Sheopuri [Li e Sheopuri 2015] afirmam que:

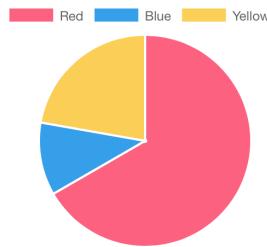


Figura 10 – Exemplo de Gráfico de Pizza utilizando Chart.js

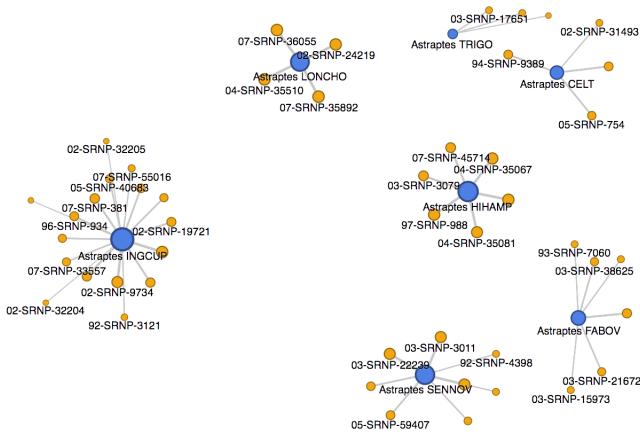


Figura 11 – Exemplo de *Network Graph* utilizando Google Charts

"Uma cor carrega um significado específico, o qual é baseado no significado aprendido ou no significado biológico"

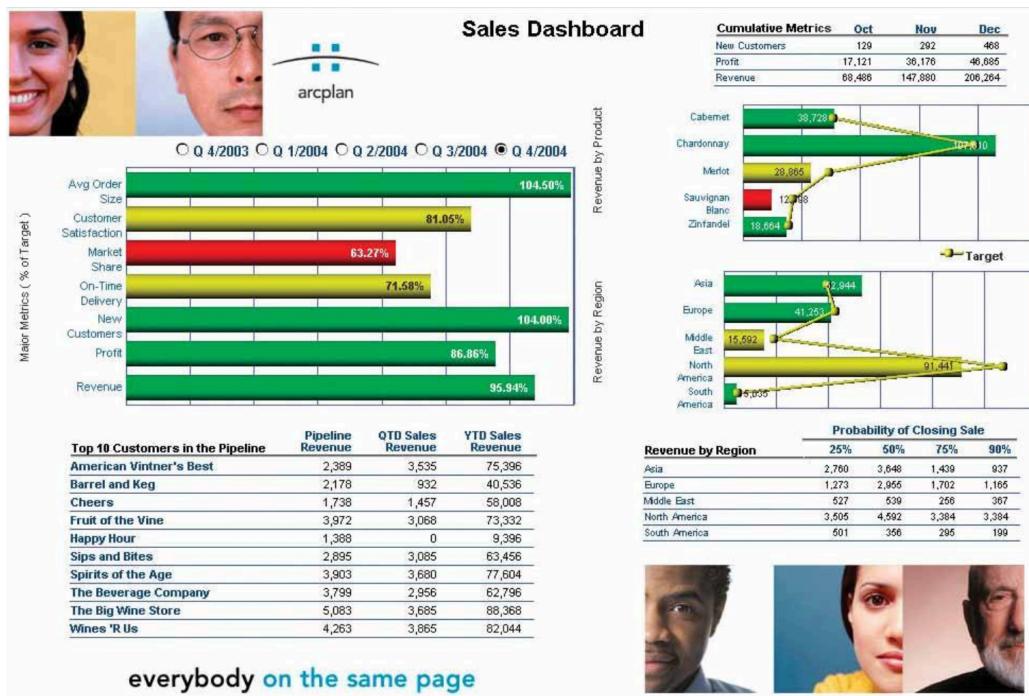
A partir desta afirmação, os autores apresentam uma Tabela 3 que mapeia uma mensagem a uma determinada cor. Utilizando-se a tabela, percebe-se que uma combinação de cores para um *dashboard* seria preto e azul por passar uma mensagem de calma, segurança e autoridade.

Em seu livro, Stephen Few [Few 2006] analisa uma variedade de *dashboards* dos quais alguns valem ser mencionados. No *dashboard* da Figura 12, o uso de *radio buttons* permite que seja feita uma seleção em relação ao período, em que se deseja analisar, porém não permite que seja feita uma comparação entre os períodos. Outro ponto a ser destacado é o uso de uma fotografia no painel. Stephen Few chama isso de *chartjunk*, pois a imagem não tem funcionalidade alguma; a não ser a de decorar o painel. Caso contrário, não é adequado o uso dessa prática.

O segundo *dashboard* (Figura 13) erra em mostrar as linhas das tabelas e dos gráficos. Essas linhas tiram o foco da informação que se quer transmitir. O uso do gráfico de pizza, neste caso, poderia ser substituído por um gráfico de barras ordenado, que comunicaria a informação de maneira mais eficiente. Um último ponto sobre este *dashboard*

Tabela 3 – Mapeamento de Cores e seus Significados

Mensagem	Cor
Aventura	Laranja
Acessibilidade	Laranja
Autoridade	Preto
Calma	Azul
Alegria	Amarelo, Laranja
Limpeza	Azul, Turquesa e Branco
Criatividade	Preto e Magenta, Azul Claro, Amarelo
Inovação	Amarelo, Roxo, Magenta
Saúde	Verde, Marrom
Paixão	Vermelho
Segurança	Azul, Marrom, Verde
Saudável	Verde Escuro, Marrom

Figura 12 – Exemplo 1 de *Dashboard* apresentado por Stephen Few. Fonte: [Few 2006]

é que ele possui muitas cores claras, o que poderia ser substituído por outras tonalidades de cor que fizessem contraste com a informação mais importante.

Por último, o *dashboard* da Figura 14 apresenta ótimas soluções para mostrar a informação quando comparado aos outros dois *dashboards*. O primeiro ponto positivo é o uso do espaço em branco para separar as seções que são apresentadas, o uso da paleta de cores também contribui para esse efeito, as únicas cores encontradas são tons de cinza, verde e dois tons de vermelho. Outro ponto positivo é que toda a informação importante se encontra em um único lugar que é no canto superior esquerdo, onde o leitor começa a leitura.

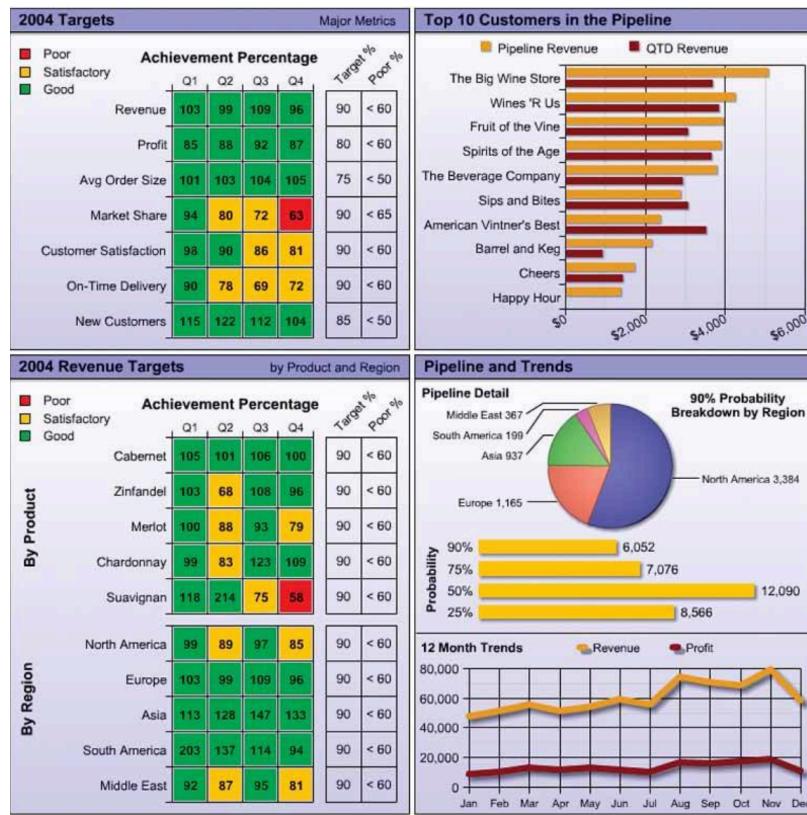


Figura 13 – Exemplo 2 de Dashboard apresentado por Stephen Few. Fonte: [Few 2006]

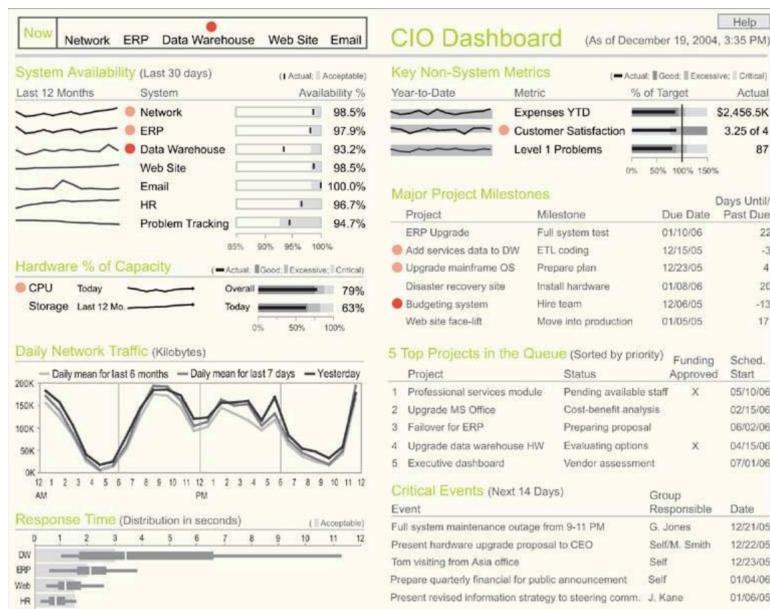


Figura 14 – Exemplo 3 de Dashboard apresentado por Stephen Few. Fonte: [Few 2006]

2.5 Resumo do Capítulo

Para que se possa tercerizar o desenvolvimento de software dentro de Órgãos Públicos, algumas regras são necessárias. Entre elas, a de que a empresa vencedora do pregão, a qual irá dispor do direito de produzir o software, entregue o mesmo com qualidade. No edital de contratação, são estabelecidos critérios de qualidade para que o software entregue seja aceito. Estes critérios de qualidade tem sua fundamentação baseada em princípios estabelecidos pela Norma SQuaRE, quanto à categoria de manutenabilidade. Essa categoria é fundamental na contratação de software, pois permite saber o quão manutenível é o software que está sendo contratado. O processo de manutenção de software está intimamente ligado a esta métrica, uma vez que quanto maior a manutenibilidade do software, menos esforço, tempo e dinheiro são gastos nesta etapa. Para definir o grau de manutenibilidade, são apresentadas suítes de métricas que avaliam conceitos específicos do software para validar se o software entregue pela tercerizada é passível de manutenção ou não. Neste trabalho, são apresentadas as suítes de Chidamber-Kemerer e MOOD. Para acompanhar a evolução da entrega e se todos os requisitos de análise estática do código estão sendo cumpridos, uma solução seria a utilização de um *dashboard* para acompanhar o desenvolvimento do software a ser entregue. Para isso, faz-se necessário o aprendizado de técnicas de visualização da informação e de conceitos para criação de um *dashboard* efetivo.

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. Dentre todas as ferramentas também é possível destacar que fora o SonarQube, Google Charts e Charts.js as demais ferramentas servem mais como suporte, podendo facilmente serem substituídas por outras ferramentas com a mesma funcionalidade.

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 GIT¹ 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 Github² 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.

¹ <https://git-scm.com/>

² <https://github.com>

3.1.3 SonarQube

O SonarQube é uma ferramenta *open-source* de análise estática de código-fonte que foca em analisar sete ramos da qualidade de código como pode ser visto na figura 15. A ferramenta 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

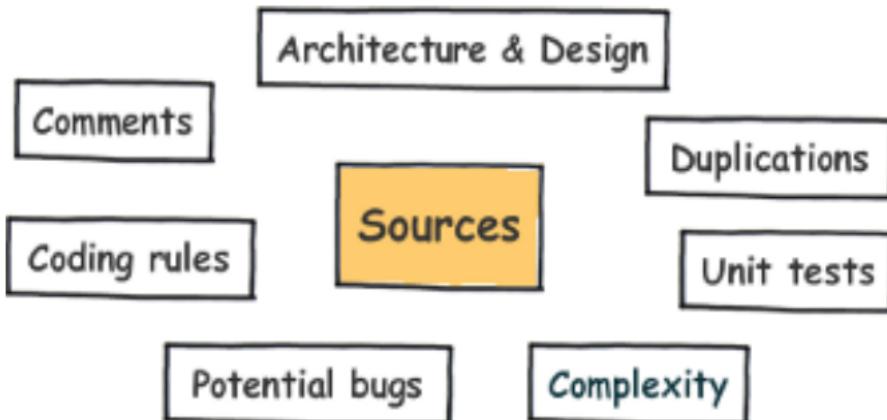


Figura 15 – Sete aspectos de qualidade de código cobertos pelo SonarQube. Fonte: [SonarQube documentation]

componentes como pode ser visto na figura 16 [SonarQube documentation].

1. **SonarQube Server** composto de três atividades principais:
 - a) Um **Web Server** para os desenvolvedores, gerentes que procuram *snapshots* da qualidade do código, e configurar uma instância 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 instância 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.

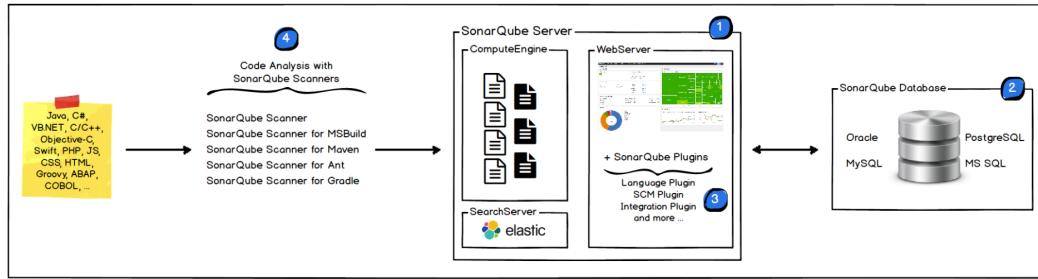


Figura 16 – Arquitetura SonarQube composta de quatro componentes: [SonarQube documentation]

O SonarQube também é facilmente integrado com outras ferramentas utilizadas durante o ciclo de vida do software. O esquema da figura 17 apresenta a implantação do SonarQube em diversos estágios do desenvolvimento de software [SonarQube documentation].

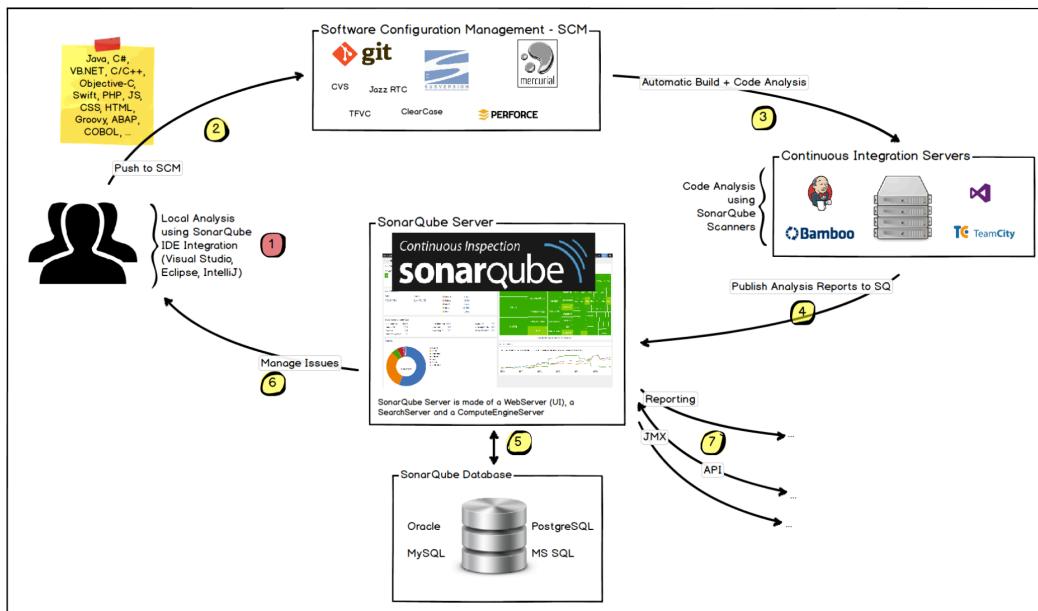


Figura 17 – Integração SonarQube em Diversas Áreas de Desenvolvimento de Software: [SonarQube documentation]

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, arrumam 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 análise estática se deu pelo fato de que grande parte dos órgãos públicos utilizam esta ferramenta para fazer a avaliação da qualidade dos softwares entregues pelas terceirizadas. 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 comum 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

Neste tópico são apresentadas as ferramentas que possuem suas funcionalidades mais voltada para o bom desenvolvimento do projeto como um todo. São ferramentas de modelagem de processo, editores de texto e gerenciamento bibliográfico.

3.2.1 Bonita

A ferramenta Bonita³ 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 <http://www.bonitasoft.com/>

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*".

3.2.3 LaTeX

O LaTeX⁴ 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 3⁵ 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

Zotero⁶ 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

Para que seja possível construir a solução o uso de ferramentas que auxiliem no processo são mais do que necessárias. As ferramentas mais importantes são o SonarQube que é o software de análise estática que é utilizado em muitos órgãos do governo, possui código-aberto e uma grande comunidade trabalhando em evoluir o software. Outra ferramenta importante é o Google Charts e o Charts.js que são responsáveis por criar os gráficos que serão utilizados no *dashbord*. As demais ferramentas funcionam muito mais como ferramentas de apoio do que ferramentas indispensáveis para a construção da solução e foram colocadas nessa seção muito mais como uma forma de documentação do material utilizado.

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 análise deste projeto serão escolhidos dois projetos, sendo ambos os projetos de código aberto extraídos do portal do software público brasileiro. A terceira e última parte do trabalho consiste em analisar o trabalho realizado através de uma avaliação qualitativa dos 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 de maneira dinâmica em que o usuário decide qual suite de métricas vai utilizar, podendo utilizar uma customizada ou utilizar uma como base (neste projeto será utilizada a do edital do DNPM [Mineral 2015]). As suites utilizadas neste trabalho são Suíte de Métricas de Chidamber-Kemerer e a Suíte MOOD.

O *dashboard* será customizável de acordo com a necessidade do órgão. O órgão poderá escolher quais métricas deseja usar e qual a faixa de valores aceitáveis para as métricas, caso ele queira, também é possível utilizar uma suíte de métricas já customizada e com valores definidos.

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 órgão.

O objetivo deste trabalho é criar uma ferramenta que auxilie na auditoria de software entregue por empresas terceirizadas, contudo percebe-se que ainda que o trabalho possua grande aprovação ele nunca substituirá 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.

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 órgãos públicos também avaliando aspectos como usabilidade e melhorias necessárias para implantação em um orgão. Para fazer esta avaliação o gestor responderá a um questionário contendo 5 perguntas referentes ao uso e funcionalidades do software produzido. O questionário é igual ao da imagem 18

4.4 Resumo do Capítulo

A proposta deste trabalho é criar uma maneira unificada de acompanhar a qualidade de código estático dos softwares entregues pela terceirizadas. Para fazer esta análise a solução utiliza da ferramenta de análise estática SonarQube e de um conjunto de métricas relacionadas às boas práticas da programação. A solução encontrada é a utilização de um dashboard que mostre o real estado de um projeto e que através de indicadores seja possível determinar a saúde do código. A última etapa deste trabalho consiste na avaliação do trabalho produzido que será feito juntamente com um gestor de projeto de um órgão público ou um professor da instituição de ensino que responderá a um questionário quanto ao software produzido.

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 18 – 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, construída 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 19 apresenta em quais características este trabalho se apresenta.

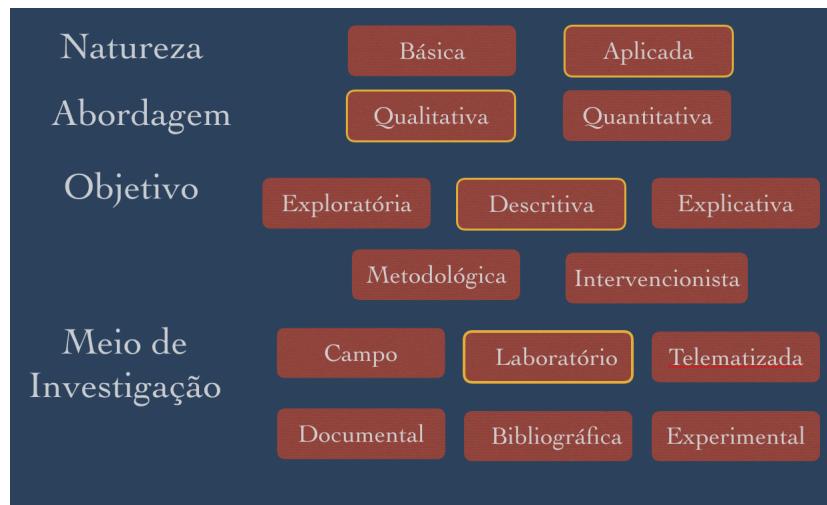


Figura 19 – Seleção das Características Metodológicas. Fonte: [Moresi e others 2003]

Este trabalho tem um caráter mais voltado para uma pesquisa aplicada por envolver características específicas 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 órgãos 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 variáveis as que compõem. Esse 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.

Outra característica deste trabalho é o fato de se tratar de uma pesquisa de laboratório. Moresi destaca que a pesquisa de laboratório atua em um ambiente controlável em que o pesquisador não tem a possibilidade de atuar em campo. Neste trabalho a pesquisa em campo se torna algo complicado pois é difícil de se conseguir acesso a órgãos públicos para instalação de uma ferramenta que ainda está em desenvolvimento. Por este motivo a pesquisa em laboratório é a mais adequada, em que são recriados as mesmas condições de uma situação em campo, porém com o controle de um ambiente simulado.

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 implementada no TCC2. A primeira fase (imagem 20) possui seis atividades principais, são elas: Definir Tema, Validar o Escopo, Elaborar um roteiro de pesquisa, Pesquisar Referências, Refinar Pesquisa e Catalogar material encontrado.

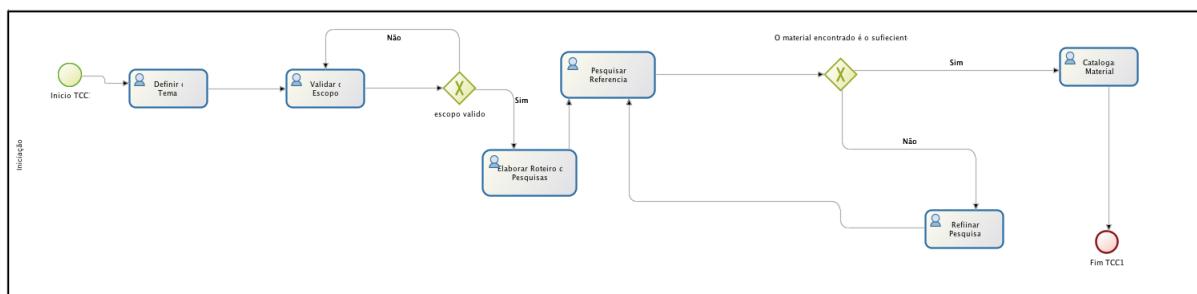


Figura 20 – Modelagem da Fase de Iniciação

A Primeira atividade de Definir um Tema acontece logo no início do processo para que seja possível estabelecer em que área e sobre o que será discutido no trabalho. Uma

vez escolhido o tema é preciso definir o escopo do trabalho para que seja possível definir quais são as fronteiras impostas tanto pelo tempo, pois é um prazo curto, quanto pelo conhecimento. Uma vez definido tema o escopo é validado juntamente com os professores orientadores para que haja um acordo entre aluno e orientador sobre o que será trabalhado.

Com o tema definido é necessário Elaborar um Roteiro de Pesquisa que consiste em encontrar a melhor string de busca para o tema. Neste projeto as strings foram montadas de acordo com o tema da pesquisa então não houve uma string geral que perpassa-se por todo o conhecimento do 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, com o decorrer o trabalho foi ganhando um escopo mais conciso.

A atividade de Elaborar Referências e Refinar Pesquisa envolviam o processo de aplicar a string nos 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 título 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. Caso o material encontrado fosse insuficiente para o assunto 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 21.

- Área 1 - Categorização por pastas dos artigos encontrados.
- Área 2 - Artigos referentes à categoria selecionada. 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. Após catalogar o material encontrado finaliza-se a fase de iniciação do projeto. Esta fase deixa como insumo para a próxima fase: o trabalho produzido até então e decisões de escopo e cenários de uso que serão implementados na fase de elaboração.

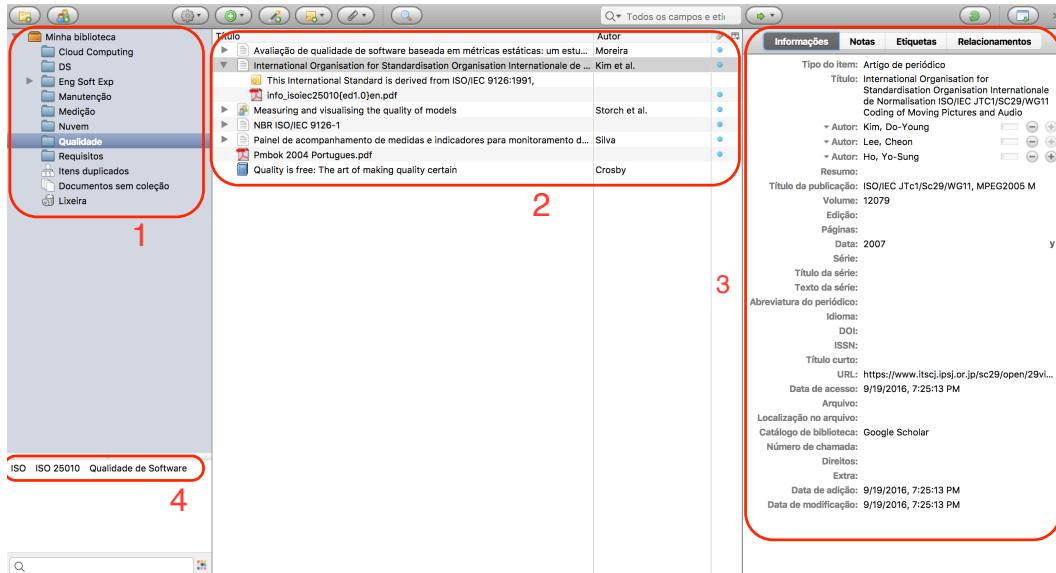


Figura 21 – Screenshot do Zotero contendo as categorias dos materiais pesquisados, suas tags e anotações

A Fase de Execução (figura 22) possui sete atividades sendo que a atividade de Documentação ocorre durante todo o processo. O objetivo desta fase é implementar o que foi decidido na fase de iniciação. A primeira atividade da segunda fase é Analisar o Ambiente. Por se tratar de um ambiente simulado é necessário que se estude quais são as melhores ferramentas e configuração entre elas para que se reproduza um ambiente mais próximo do real possível. Uma vez definido é necessário Configurar este ambiente o que deve ser feito seguindo as configurações de um ambiente real.

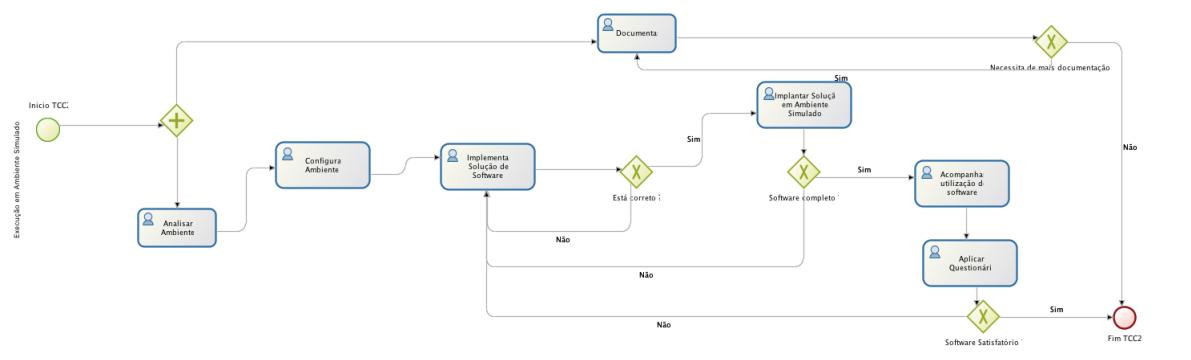


Figura 22 – Modelagem da Fase de Execução

A terceira atividade é Implementar a Solução. Esta atividade segue os princípios de desenvolvimento ágil para construção de soluções em software, para isso seu desenvolvimento é feito de maneira iterativa incremental de forma que a cada iteração haja um implemento funcional de software. Ao fim de cada iteração se o software produzido estiver funcionando e testado ele é colocado no ambiente simulado para que se possa acompanhar o seu funcionamento enquanto em um ambiente "produção".

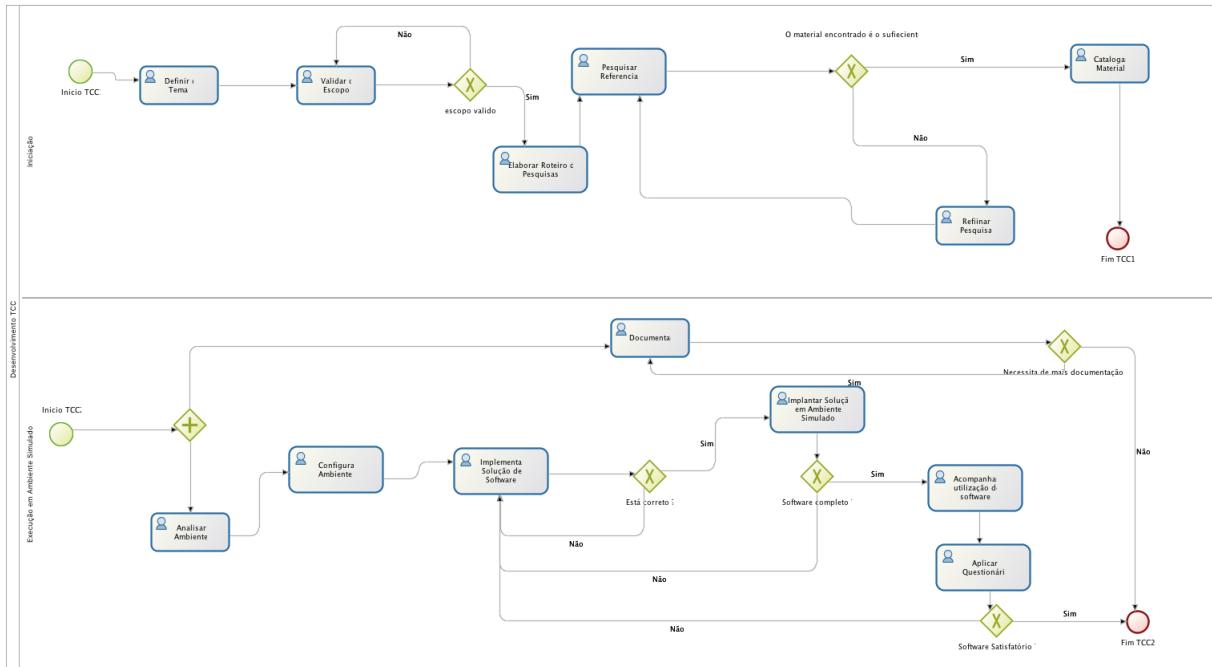


Figura 23 – Plano de Pesquisa

Uma vez que a solução em software esteja completa ela é disponibilizada para um conjunto de utilizadores que irão testar a solução e documentar as suas impressões quanto a utilização da solução. A última atividade desta fase é relacionada ao questionário que é aplicado aos utilizadores da ferramenta para que se possa avaliar o quanto efetiva foi a solução.

Com a Fase de Execução finalizada o processo se encerra. Nesta segunda fase os principais artefatos são: a solução funcionando e um refinamento do documento entregue na primeira fase. A figura 23 apresenta uma visão geral do processo que será aplicado neste trabalho.

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 podem ser associados a 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 24 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].

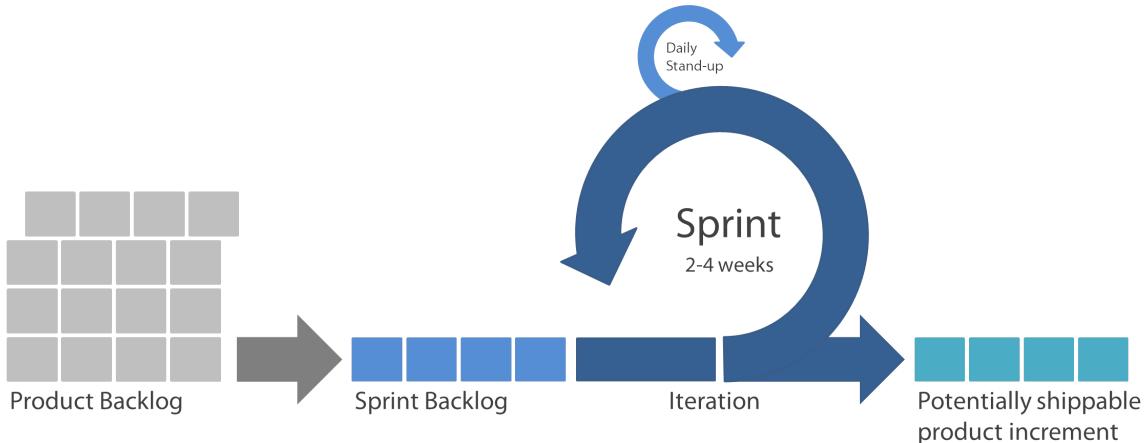


Figura 24 – Ciclo de Desenvolvimento do Scrum

Os principais conceitos que foram utilizados da metodologia foram:

- **Product Backlog:** Lista de atividades que representam as funcionalidades que serão construídas 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 25. E a confirmação ou critérios de aceitação é um *checklist* abordando o que deve ser verificado para que a história seja dada como concluída.
- **Story Point:** Unidade relativa que caracteriza o esforço da equipe de desenvolvimento para finalizar uma atividade. A escala utilizada é definida pela equipe de

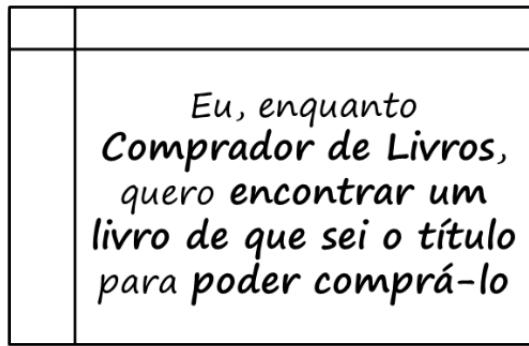


Figura 25 – Exemplo de História de Usuário. Fonte: [Sabbagh 2014]

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 26.

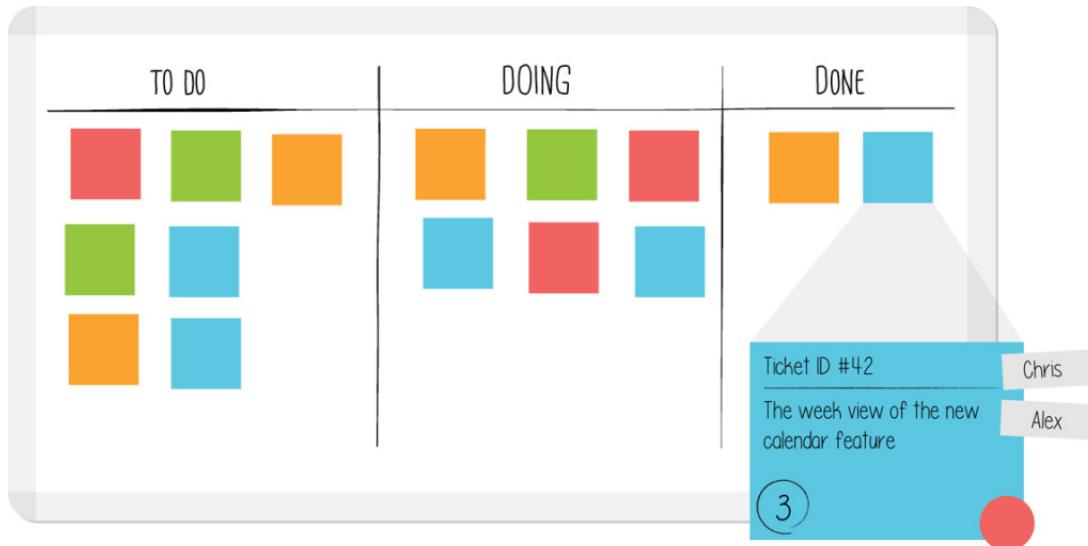


Figura 26 – Exemplo de *Kanban*

5.3 Cronograma

Para que se possa ter uma visão mais abrangente da organização do trabalho foi criado um cronograma em que constam as atividades definidas no 5.1.1. Este cronograma serve como orientação ao desenvolvimento e planejamento do projeto contudo com o decorrer das atividades este artefato pode sofrer alterações. O cronograma foi dividido em duas tabelas referentes às atividades do TCC1 e TCC2.

5.3.1 Cronograma TCC1

A tabela 4 apresenta um cronograma em relação as atividades a serem desenvolvidas enquanto TCC1

Tabela 4 – Cronograma TCC1

Atividade	Agosto	Setembro	Outubro	Novembro
Definir Tema	X			
Validar Escopo	X			
Elaborar Roteiro de Pesquisa		X	X	
Pesquisar Referência			X	X
Refinar Pesquisa			X	X
Catalogar Material				X

5.3.2 Cronograma TCC2

A tabela 5 apresenta um cronograma em relação as atividades a serem desenvolvidas enquanto TCC2

Tabela 5 – Cronograma TCC2

Atividade	Março	Abril	Maio	Junho
Documentar	X	X	X	X
Analizar Ambiente	X			
Configurar Ambiente	X			
Implementar Solução de Software		X	X	
Implantar Solução em Ambiente Simulado		X	X	
Acompanhar Utilização do Software				X
Aplicar Questionário				X

5.4 Resumo do Capítulo

A pesquisa pode ser classificada de diversas formas, neste a projeto a natureza da pesquisa pode ser classificada como Aplicada devido ao uso nas áreas de TI. Quanto a sua abordagem ela é do tipo Qualitativa pois a forma de se analisar e coletar os dados é feito de maneira empírica e através da observação. Sobre o seu objetivo essa pesquisa tem um caráter Descritivo pois ela observa fatores de um grupo e os descreve neste caso a maneira como funciona a aquisição de software por parte da APF. E por último quanto ao meio de investigação que Laboratorial pois todo o desenvolvimento da pesquisa é feita em um ambiente controlado e não em campo. Quanto a metodologia de desenvolvimento optou-se por uma adaptação da metodologia Scrum, isto se deu pelo fato de que o framework da metodologia é adaptável para times de desenvolvimento pequenos e com entregas de software em curtos períodos de tempo.

6 Resultados Parciais

Neste capítulo estão descritos os resultados que foram obtidos até a apresentação do TCC1. Ele se divide em duas seções, a seção 6.0.1 que apresenta o estado do projeto atual e a seção 6.0.2 que aborda quais serão os próximos passos no desenvolvimento do trabalho.

6.0.1 *Status* atual do Projeto

Como havia sido definido no capítulo de introdução, este trabalho tem como objetivo principal desenvolver uma solução em software para monitoramento da qualidade de código. Tomando esse objetivo como linha guia para o desenvolvimento e elaboração deste projeto as primeiras atitudes tomadas foram aprofundar mais o conhecimento do assunto e garantir que será possível importar as métricas extraidas pelo SonarQube.

A aquisição de conhecimento foi feita através de um levantamento bibliográfico em várias bases de pesquisa e que influenciaram na escrita do capítulo de Referências Bibliográficas e na tomada de algumas decisões acerca do trabalho, decisões como a escolha da suíte de métricas e as ferramentas para plotagem dos gráficos.

Quanto à importação das métricas a forma a ser utilizada será de leitura da tela. Foi criado um script em python que faz a leitura das informações da tela e separa elas por métricas e por projeto. Esse script será responsável por coletar e identificar as métricas a serem escolhidas pelo gestor de projetos.

A tabela 6 mostra a completude do trabalho desenvolvido até o momento, separando por atividades e os meses em que se estima que essas atividades ocorram. Pela tabela é possível perceber que todas as atividades que haviam sido planejadas para o TCC1 já foram concluídas e a atividade de Analisar Ambiente já foi iniciada devido a instalação do SonarQube. A atividade de Documentação se encontra em 50% pois se refere à documentação entregue enquanto TCC1.

6.0.2 Próximas Atividades

Uma vez que já se consegue extrair as informações do SonarQube a próxima etapa é a de construir o *dashboard*. Para isso será levantado todas as funcionalidades que o *dashboard* deve apresentar. Este conjunto de atividades irá compor o *product backlog*. Com o *product backlog* definido, é necessário que se faça a divisão das sprints de acordo com o número funcionalidades. Todas essas atividades fazem parte das atividades necessárias para construção do software.

Tabela 6 – *Status* Completude do Trabalho

Atividade	Status
Definir Tema	100%
Validar Escopo	100%
Elaborar Roteiro de Pesquisa	100%
Pesquisar Referência	100%
Refinar Pesquisa	100%
Catalogar Material	100%
Documentar	50%
Analisar Ambiente	20%
Configurar Ambiente	0%
Implementar Solução de Software	0%
Implantar Solução em Ambiente Simulado	0%
Acompanhar Utilização do Software	0%
Aplicar Questionário	0%

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 45.
- 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 48.
- 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.
- DOUMONT, J.-L.; VANDENBROECK, P. Choosing the right graph. *IEEE transactions on professional communication*, v. 45, n. 1, p. 1–6, 2002. Disponível em: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=988358>. Citado na página 39.
- FERENC, R. et al. Source Meter Sonar Qube Plug-in. In: . [S.l.]: IEEE, 2014. p. 77–82. Citado na página 46.
- FEW, S. *Information Dashboard Design - The effective Visual Comunication of Data*. [S.l.]: O'Reilly, 2006. Citado 5 vezes nas páginas 13, 38, 40, 41 y 42.
- 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 na página 55.
- GIL, A. C. *Como elaborar projetos de pesquisa*. São Paulo (SP): Atlas, 2002. OCLC: 817765297. ISBN 978-85-224-3169-4. Citado na página 56.
- 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, 36 y 37.

GOOGLE Charts documentation. <<https://developers.google.com/chart/interactive/docs/reference>>. Accessed: 2016-10-10. Citado na página 48.

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 38.

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.I.]. Citado 2 vezes nas páginas 13 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.I.: s.n.], 2001. p. 667–670. ISSN 1063-6773. Citado na página 32.

LI, Y.; SHEOPURI, A. Creative design of color palettes for product packaging. In: *2015 IEEE International Conference on Multimedia and Expo (ICME)*. IEEE, 2015. p. 1–6. Disponível em: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=7177443>. Citado na página 39.

LOPEZ-MARTINEZ, J. et al. Problems in the Adoption of Agile-Scrum Methodologies: A Systematic Literature Review. In: . IEEE, 2016. p. 141–148. ISBN 978-1-5090-1074-5. Disponível em: <<http://ieeexplore.ieee.org/document/7477924/>>. Citado na página 59.

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://ecomam.ktu.lt/index.php/elt/article/view/372>>. Citado na página 60.

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 45.

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 35.

MINERAL, D. N. de P. *Edital Pregão Eletronico No 14/2015*. [S.l.], 2015. Citado 4 vezes nas páginas 15, 27, 51 y 52.

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 35.

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, 55 y 56.

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*. AISTI, 2016. p. 1–6. Disponível em: <http://ieeexplore.ieee.org/abstract/document/7521555>. Citado na página 59.

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 3 vezes nas páginas 13, 32 y 33.

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

PRESSMAN, R. S. *Software Engineering - A Practitioner'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%A7%C3%B5es.+Hoje+%C3%A9+poss%C3%ADvel+ver+os+assuntos+ligados%22+%22Mas,+segundo+o+pr%C3%BDrio+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 14, 60 y 61.

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 30.

SCHWENDIMANN, B. et al. Perceiving learning at a glance: A systematic literature review of learning dashboard research. *IEEE Transactions on Learning Technologies*, p. 1–1, 2016. ISSN 1939-1382. Disponível em: <<http://ieeexplore.ieee.org/document/7542151/>>. Citado 2 vezes nas páginas 13 y 38.

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 23.

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 38.

SOMMERVILLE, 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, 46 y 47.

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