



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-

78 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 Estática de Software. 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

**Prof. Dr. Luiz Carlos Miyadaira
Ribeiro Jr**
Convidado 1

Prof. Dr. André Barros de Sales
Convidado 2

Brasília, DF
2016

"A vida passa rápido demais; e se você não parar de vez em quando para vive-la, acaba perdendo seu tempo" Ferris Bueller

Agradecimentos

Agradeço primeiramente a Deus e tenho plena certeza de que tudo o que já alcancei e ainda vou alcançar é por causa do seu amor por mim. Agradeço também aos meus pais Claudio e Milene que sempre batalharam e abriraram mão de muitos prazeres para que eu pudesse viver os meus sonhos. Agradeço às minhas irmãs Marianna e Débora por sempre me apoiarem com palavras de incentivo e as em alguns momentos de correção. Agradeço ao meu orientador Maurício Serrano e minha coorientadora Milene Serrano que sempre me apoiaram, defenderam e corrigiram.

Resumo

A contratação de software é uma prática comum dentro dos Órgão Públicos Brasileiros. No momento que a empresa ganha a licitação de um serviço, a empresa passa a ter o direito de contratação, para prestar o serviço ao Órgão contratante. Este trabalho tem por objetivo propor um *dashboard* que, com a ajuda de recursos visuais, seja uma ferramenta que auxilie no processo de contratação de software em Órgãos Públicos. Para esta solução, o *dashboard* funciona como uma ferramenta de visualização de indicadores, determinados pelo Gestor de Projetos do Órgão Público. O *dashboard* exibe os indicadores dos projetos que estão sendo desenvolvidos pela Contratante para o Gestor. O trabalho consistiu de duas fases: Iniciação e Execução. O objetivo da primeira etapa é estabelecer fundamentos (teóricos e arquiteturais) para a fase de Execução, enquanto o foco da segunda fase está na implementação da solução. Optou-se pelo uso do Scrum como metodologia de desenvolvimento ágil, contudo algumas mudanças foram necessárias, principalmente devido ao escopo e ao tamanho da equipe. Os resultados alcançados até o momento, estão mais voltados para a coleta das métricas utilizando o software SonarQube, e a exibição, por meio de gráficos, dos dados em uma página web.

Palavras-chaves: dashboard. qualidade. contratação de software. monitoramento de métricas. visualização de indicadores.

Abstract

The contracting of software is a common practice within the Brazilian Public Organs. At the moment the company wins the bid for a service, the company will have the right to contract, to render the service to the contracting entity. This work aims to propose a dashboard that, with the help of visual resources, is a tool that helps in the process of contracting software in Public Bodies. For this solution, dashboard works as a tool for visualizing indicators, as determined by the Public Organ Project Manager. The dashboard displays the indicators of the projects being developed by the Employer to the Manager. The work consisted of two phases: Initiation and Execution. The objective of the first step is to establish fundamentals (theoretical and architectural) for the Execution phase, while the focus of the second phase is on the implementation of the solution. We chose to use Scrum as an agile development methodology, but some changes were necessary, mainly due to the scope and size of the team. The results achieved so far are more focused on collecting the metrics using the SonarQube software, and displaying, graphically, the data on a web page.

Key-words: dashboard. quality. hiring software. monitoring metrics. indicators.

Lista 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 Frecente. 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]	43
Figura 15 – Aplicação de Visibilidade do Sistema. Fonte: [Selvaraj]	43
Figura 16 – Combinação de Termos do Mundo Real é Melhor Aceito do Que Termos Técnicos. Fonte: [Selvaraj]	44
Figura 17 – Liberdade ao Usuário Poder Interromper ou Alterar Uma Ação Feita no Sistema. Fonte: [Selvaraj]	44
Figura 18 – Seguir os Padrões é Melhor do Que Propor Algo Diferente. Fonte: [Selvaraj]	45
Figura 19 – Prevenir Erros Antes Que Eles Ocorram. Fonte: [Selvaraj]	45
Figura 20 – Reconhecer é Melhor do Que Lembrar. Fonte: [Selvaraj]	46
Figura 21 – Uso de "Aceleradores"Para Usúarios Mais Experientes. Fonte: [Selvaraj]	46
Figura 22 – Exemplo Mostrando Telas de Erro de Maneira a Tranquilizar o Usuário. Fonte: [Selvaraj]	47
Figura 23 – Uso de Documentação Mesmo Para Pequenas Tarefas. Fonte: [Selvaraj]	48

Figura 24 – Sete aspectos de qualidade de código cobertos pelo SonarQube. Fonte: [SonarQube documentation]	50
Figura 25 – Arquitetura SonarQube composta de quatro componentes: [SonarQube documentation]	51
Figura 26 – Integração SonarQube em Diversas Áreas de Desenvolvimento de Software: [SonarQube documentation]	51
Figura 27 – Processo de Desenvolvimento Adotado Pelo Órgão X. Fonte: [Schaidt e Regis 2014]	56
Figura 28 – Ciclo de Verificação Utilizando a Solução Compartilhada Entre Órgão Contratante e Empresa Contratada	56
Figura 29 – Tela Login	58
Figura 30 – Tela Home	59
Figura 31 – Tela Dashboard	60
Figura 32 – Tela Adicionar	60
Figura 33 – Questionário a ser aplicado ao fim do período de teste	62
Figura 34 – Seleção das Características Metodológicas. Fonte: [Moresi 2003]	63
Figura 35 – Modelagem da Fase de Iniciação	65
Figura 36 – <i>Screenshot</i> do Zotero contendo as categorias dos materiais pesquisados, suas <i>tags</i> e anotações	66
Figura 37 – Modelagem da Fase de Execução	67
Figura 38 – Plano de Pesquisa	68
Figura 39 – Ciclo de Desenvolvimento do Scrum	69
Figura 40 – Exemplo de História de Usuário. Fonte: [Sabbagh 2014]	70
Figura 41 – Exemplo de <i>Kanban</i>	70

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	71
Tabela 5 – Cronograma TCC2	71
Tabela 6 – <i>Status</i> Completude do Trabalho	74

Listas de abreviaturas e siglas

APF	Administração Pública Federal
API	<i>Application Programming Interface</i>
CBO	<i>Coupling Between Objects</i>
DIT	<i>Depth of Inheritance</i>
DNPM	Departamento Nacional de Produção Mineral
ISO	<i>International Organization for Standardization</i>
LCOM	<i>Lack of Cohesion in Methods</i>
MIT	<i>Massachusetts Institute of Technology</i>
NOC	<i>Number of Children</i>
RFC	<i>Response for a class</i>
SCM	<i>Software Configuration Management</i>
TCC	Trabalho de Conclusão de Curso
TI	Tecnologia da Informação
TCU	Tribunal de Contas da União
UI	<i>User Interface</i>
WDC	<i>Weighted Methods per Class</i>

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	Dez Heurísticas de Usabilidade Propostas por Nielsen	42
2.6	Resumo do Capítulo	47
3	SUPORTE TECNOLÓGICO	49
3.1	Ferramentas de Programação	49
3.1.1	GIT	49
3.1.2	Github	49
3.1.3	SonarQube	50
3.1.4	Google Charts e Charts.js	52
3.2	Ferramentas de Gerenciamento	52
3.2.1	Bonita	53
3.2.2	Mac OS X	53
3.2.3	LaTeX	53
3.2.4	Sublime Text 3	53

3.2.5	Zotero	53
3.3	Resumo do Capítulo	53
4	PROPOSTA	55
4.1	Ambiente Simulado	55
4.2	Coleta das Métricas	57
4.3	Criação do <i>Dashboard</i>	57
4.4	Avaliação	59
4.5	Resumo do Capítulo	61
5	METODOLOGIA	63
5.1	Metodologia de Pesquisa	63
5.1.1	Plano Metodológico	64
5.2	Metodologia de Desenvolvimento	68
5.3	Cronograma	71
5.3.1	Cronograma TCC1	71
5.3.2	Cronograma TCC2	71
5.4	Resumo do Capítulo	72
6	RESULTADOS PARCIAIS	73
6.0.1	<i>Status</i> Atual do Projeto	73
6.0.2	Próximas Atividades	73
	REFERÊNCIAS	75

1 Introdução

Neste primeiro capítulo, é apresentada uma visão mais ampla do trabalho. Tal visão objetiva 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. Uma vez contextualizado, 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 (TCU). 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 [Schnaider 2004].

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" [Crosby 1980]. 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 e Regis 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. Além dos resultados abordados, também espera-se alcançar maior conhecimento na área de Qualidade de Software também como nas áreas associadas a esta, por exemplo, Gerência de Configuração, Integração Contínua e Controle de Versão.

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** serve como base bibliográfica para conceituação de termos que são usados ao longo do 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 terceirização.

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 1993]. 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 [TCU 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: taxa de cobertura de código, complexidade por método e LCOM4, as quais não possuem valores definidos de cobrança [Mineral 2015], 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]

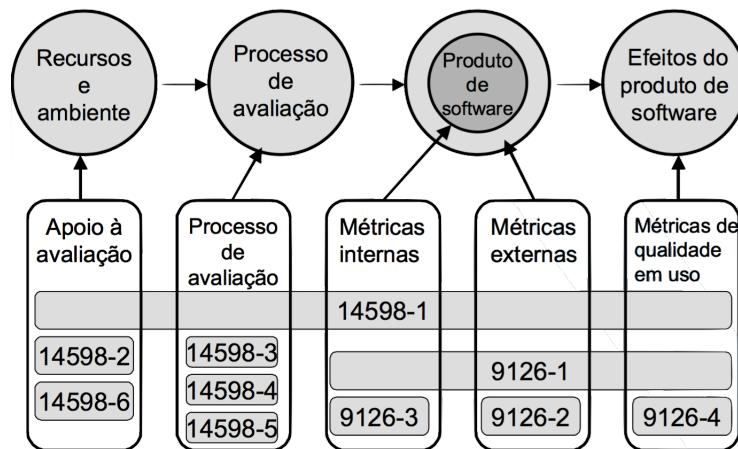


Figura 1 – Relação entre as NBR ISO/IEC 9126 e NBR ISO/IEC 14598 .Fonte: [NBR ISO/IEC 9126-1 2016]

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:

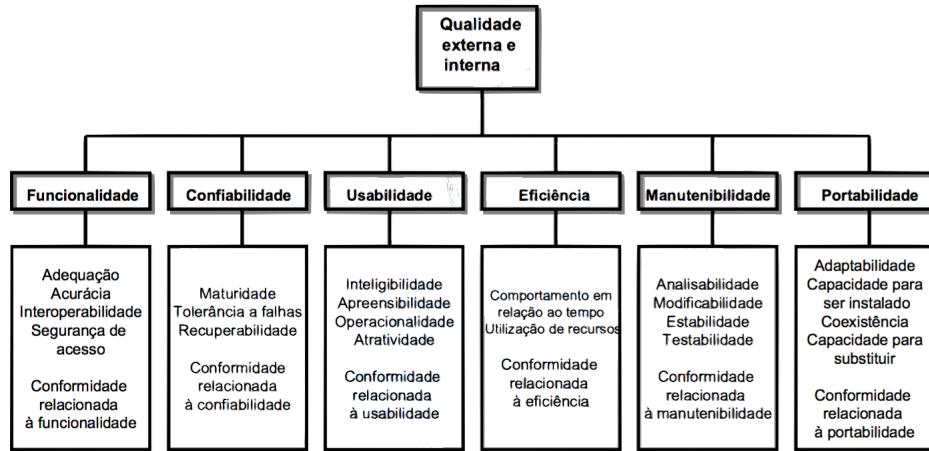


Figura 2 – Modelo de Qualidade para Qualidade Interna e Externa .Fonte: [NBR ISO/IEC 9126-1 2016]

- **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 e Regis 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.

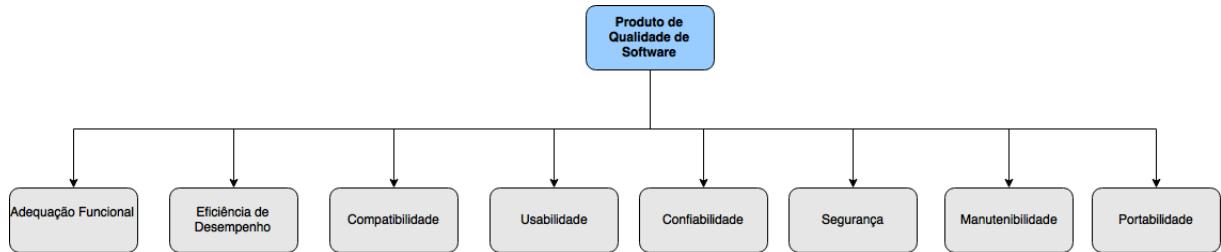


Figura 3 – Produto de Qualidade de Software. Fonte: [ISO 25010]

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

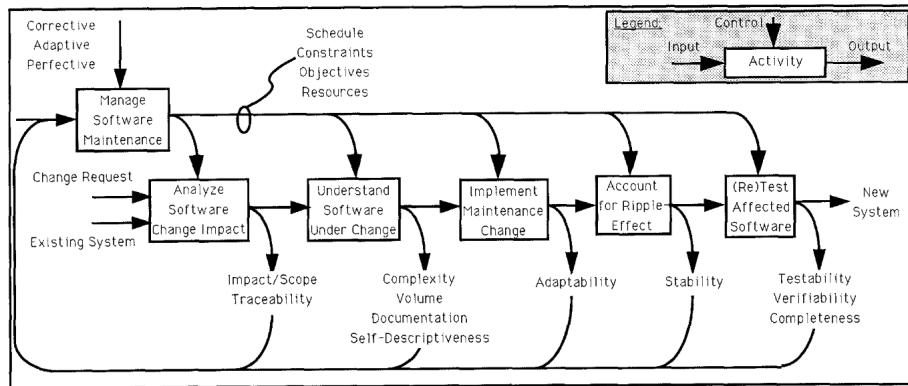


Figura 4 – Diagrama das Atividades de Manutenção de Software. Fonte: [Pfleeger e Bohner 1990]

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.

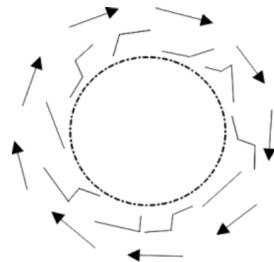


Figura 5 – Exemplo de glifo utilizado na visualização da variação das medidas relacionadas com o desenvolvimento computacional. Fonte: [Gomes e Tavares 2011]

- **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.

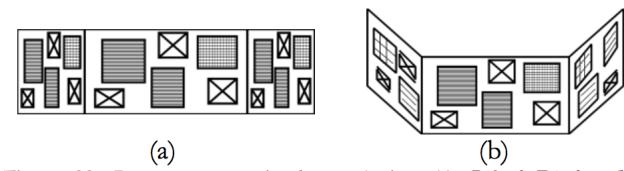


Figura 6 – Comparativo entre a) Bifocal Display e b) Perspective Wall

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* [Few 2006]. 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).

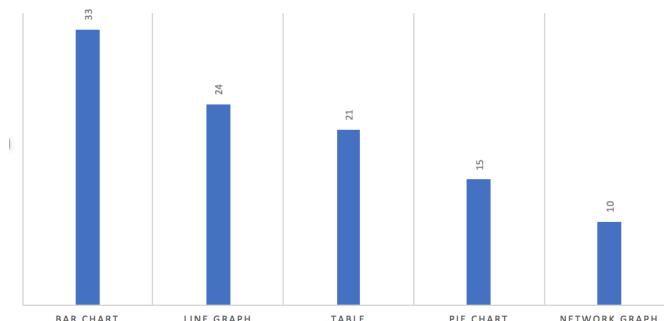


Figura 7 – Tipos de Visualização Mais Frequente. Fonte: [Schwendimann et al. 2016]

- 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, Philippe 2002]

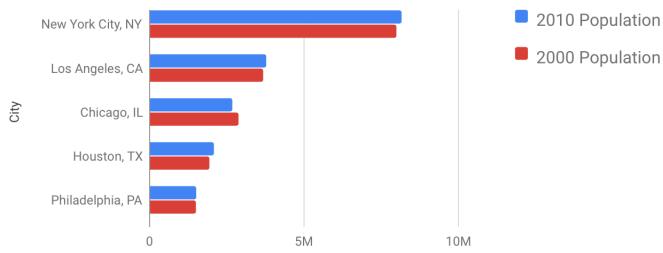


Figura 8 – Exemplo de Gráfico de Barra Utilizando Google Charts

- 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.

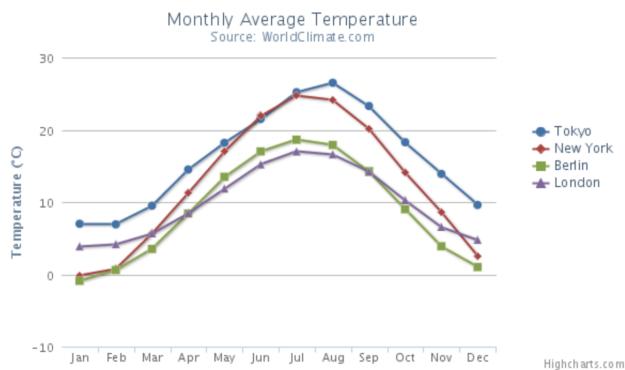


Figura 9 – Exemplo de Gráfico de Linhas utilizando HiCharts

- 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:

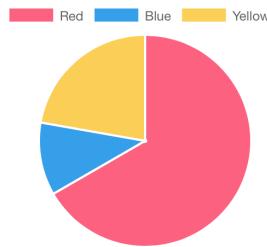


Figura 10 – Exemplo de Gráfico de Pizza utilizando Chart.js

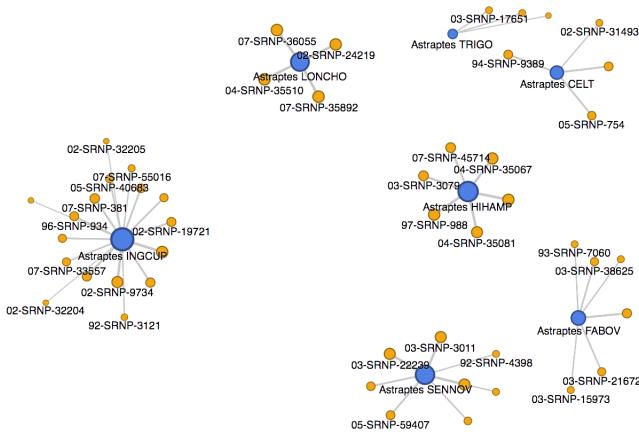


Figura 11 – Exemplo de *Network Graph* utilizando Google Charts

"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

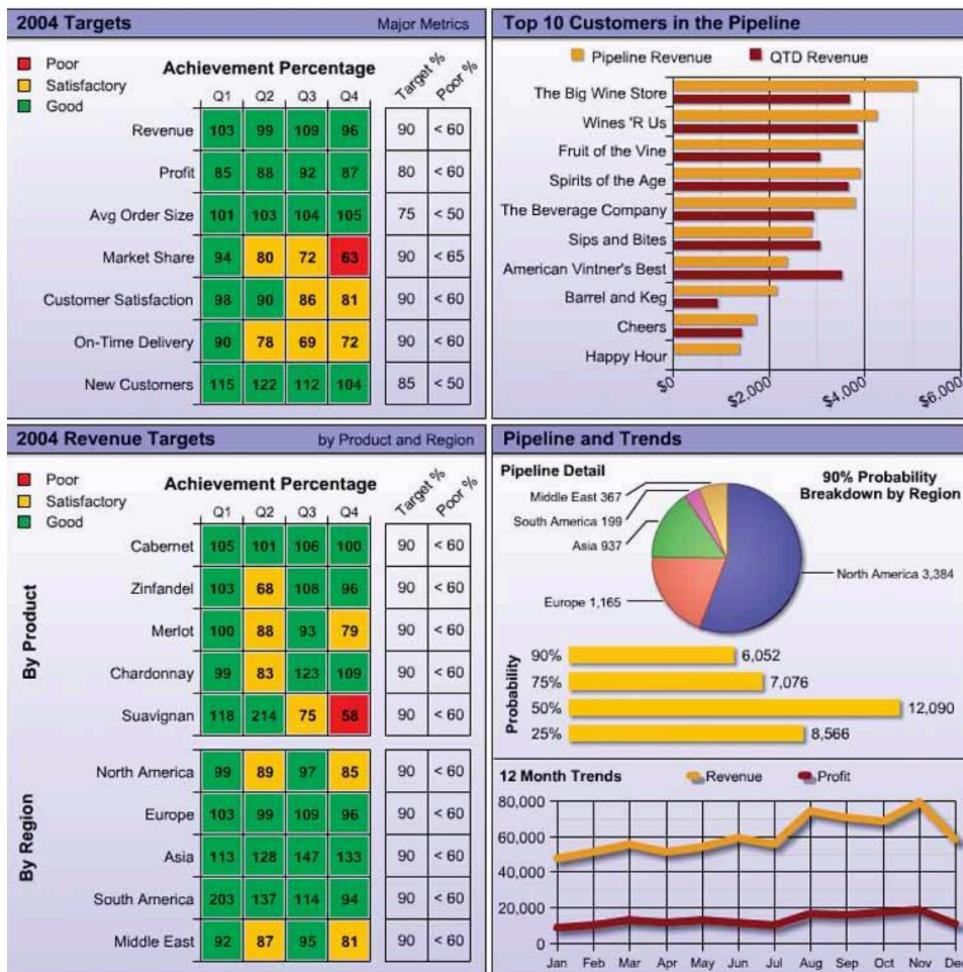


Figura 13 – Exemplo 2 de *Dashboard* apresentado por Stephen Few. Fonte: [Few 2006]

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.

O *dashboard* é uma ferramenta muito útil na visualização de dados, quando utilizada da maneira correta. Nielsen apresenta um conjunto de dez práticas que, quando aplicadas de maneira correta, ajudam na usabilidade de uma aplicação. O papel da usabilidade é fazer com que a interação humano-computador ocorra de maneira amigável e eficiente.

2.5 Dez Heurísticas de Usabilidade Propostas por Nielsen

Segundo a ISO 25010, usabilidade é definido como sendo o grau em que um produto ou sistema pode ser usado por determinados grupo de usuários para atingir os objetivos especificados [ISO 25010]. Jacob Nielsen afirma que existe um conjunto de dez boas práticas [Nielsen]. Este conjunto de práticas também conhecidas como Heurísticas de Nielsen

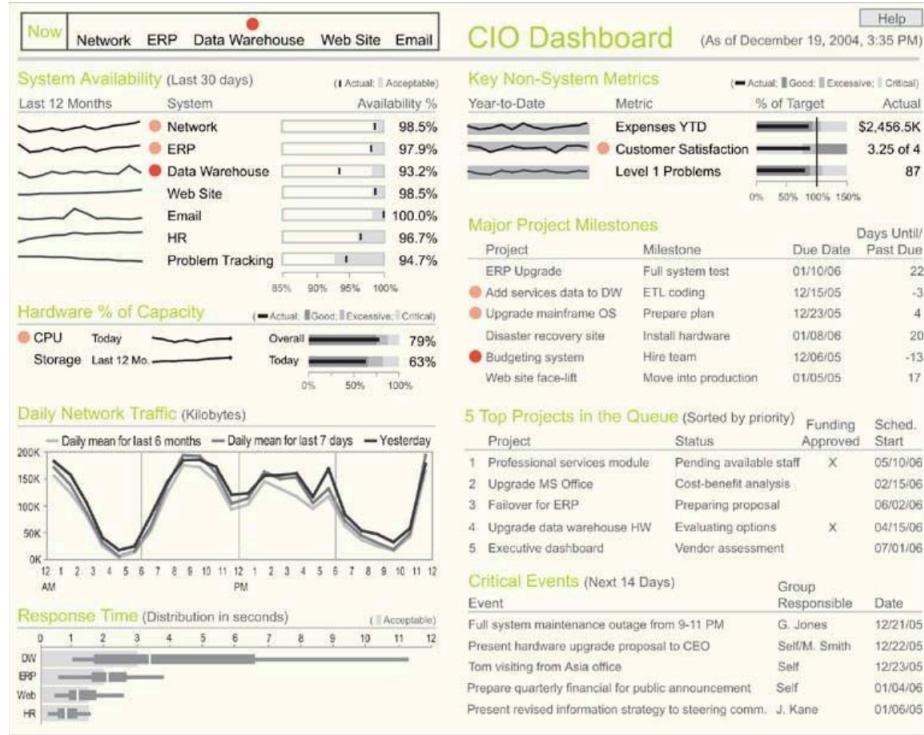


Figura 14 – Exemplo 3 de *Dashboard* apresentado por Stephen Few. Fonte: [Few 2006]

(o termo heurística, se deve ao fato de que o objetivo é a natureza da regra e o que ela significa em um contexto geral, ao invés de regras específicas e imutáveis). As boas práticas são:

1. **Visibilidade de *Status* do Sistema:** o sistema deve sempre manter o usuário informado sobre o que está acontecendo, através de um *feedback* e um tempo considerável. Um exemplo, na Figura 15 é possível perceber que a cor do ícone do avião, indica se um vôô está atrasado ou não. Na mesma Figura é possível ver um *status* na barra superior indicando que uma mensagem está sendo enviada.

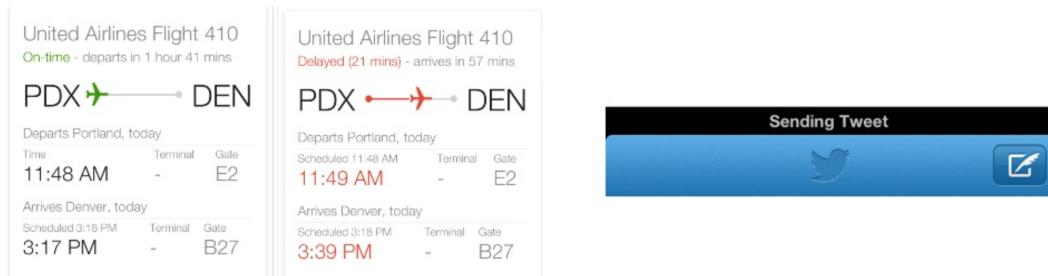


Figura 15 – Aplicação de Visibilidade do Sistema. Fonte: [Selvaraj]

2. **Mesclar Ações do Sistema Com o Mundo Real:** o sistema deve falar a linguagem do usuário. O uso de termos comuns, próximos a realidade do usuário é melhor

aceito, do que termos mais técnicos. A Figura 16 apresenta dois exemplos de uso desta técnica. No primeiro caso, o uso de termos comuns e mais informais, facilita o entendimento da ação que o usuário está tomando. No segundo caso, há uma aproximação com o mundo real, em que o termo FAQ (*Frequently Asked Questions*) que não é um termo tão conhecido, foi substituído por "How can we help you?", que é um termo muito mais utilizado no cotidiano do usuário.

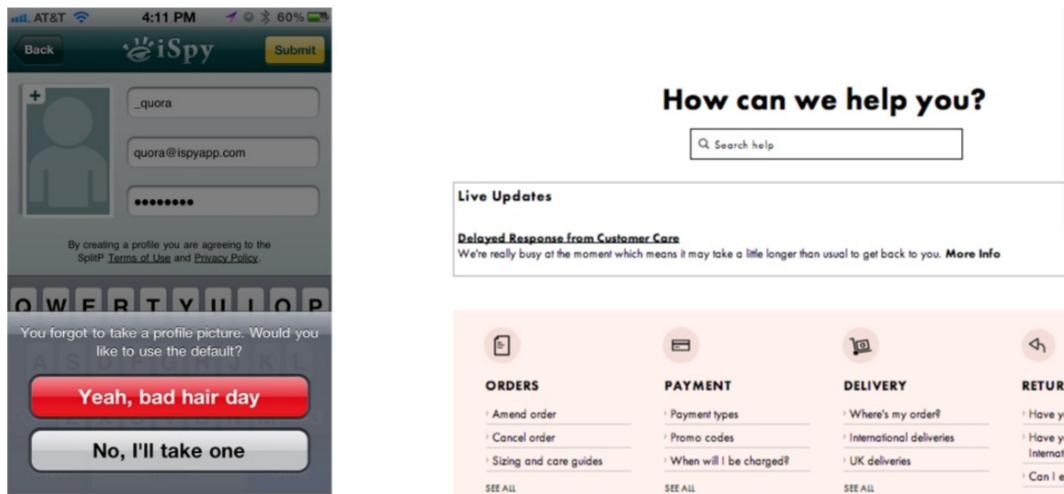


Figura 16 – Combinação de Termos do Mundo Real é Melhor Aceito do Que Termos Técnicos. Fonte: [Selvaraj]

- 3. Controle do Usuário e Liberdade:** é comum que o usuário escolha uma função do sistema sem querer, é necessário que o sistema tenha uma "saída de emergência". Esta prática pode ser observada na Figura 17 em que ao enviar uma mensagem o sistema exibe a opção de cancelar o envio.



Figura 17 – Liberdade ao Usuário Poder Interromper ou Alterar Uma Ação Feita no Sistema. Fonte: [Selvaraj]

- 4. Consistências e Padrões:** o usuário não deve se preocupar se uma palavra, situação ou ação significam a mesma coisa. É recomendável que se siga a convenção da plataforma. A Figura 18 apresenta três cabeçalhos de *websites* que utilizam a ação de realizar o *login*, no canto superior direito da página, pois este é o local convencional. Quando um padrão é quebrado, pode causar um incômodo ao usuário.

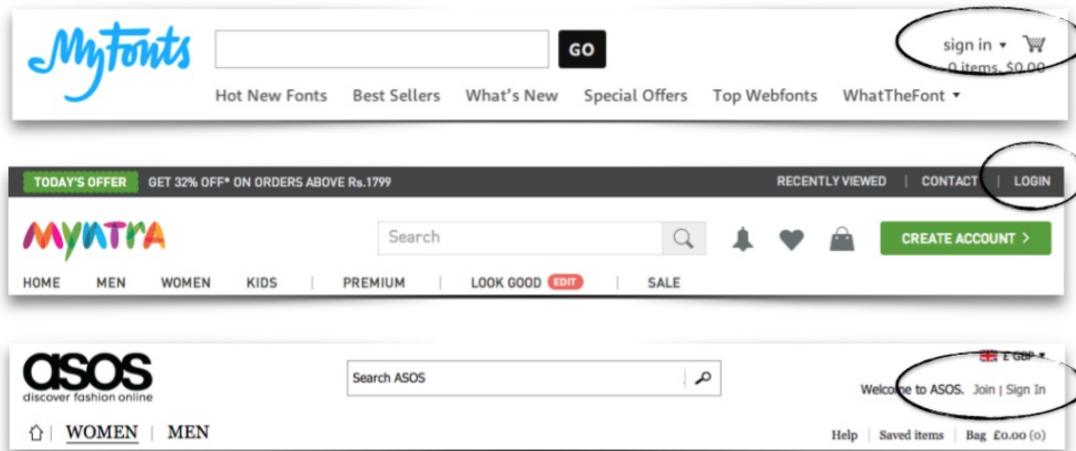


Figura 18 – Seguir os Padrões é Melhor do Que Propor Algo Diferente. Fonte: [Selvaraj]

5. Prevenção de Erros: melhor do que mensagens de erro, é ter o cuidado e prevenir para que esses erros não ocorram. É recomendável que exista uma opção de confirmação para o usuário antes dele realizar a ação. A Figura 19 apresenta um *pop-up* que é apresentado ao usuário quando ele deseja realizar a exclusão de um arquivo. Esta prática previne o usuário de realizar ações críticas por acidente, ou por ingenuidade.

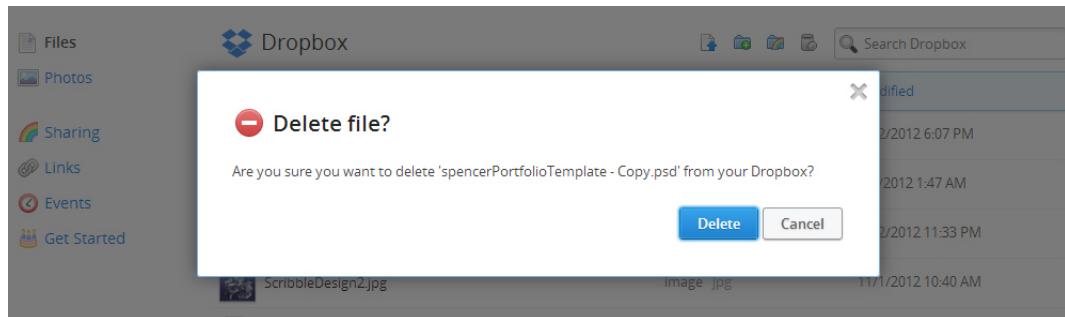


Figura 19 – Prevenir Erros Antes Que Eles Ocorram. Fonte: [Selvaraj]

6. Reconhecer ao Invés de Lembrar: instruções de uso do sistema devem ser visíveis e fáceis de se recuperar quando necessário. A Figura 20 apresenta uma página em que facilmente o usuário reconhece a funcionalidade da página. A utilização de apenas um botão para realizar a ação, faz com que distrações sejam menos aparentes.

7. Flexibilidade e Eficiência de Uso: utilização de "aceleradores", que são invisíveis para usuário inexperientes, mas que agilizam tarefas para os usuários mais experientes. O exemplo apresentado na Figura 21, mostra duas situações em que são colocados funcionalidades para um usuário mais experiente. Na primeira imagem existe a opção por filtros de busca, para se obter um categoria específica de

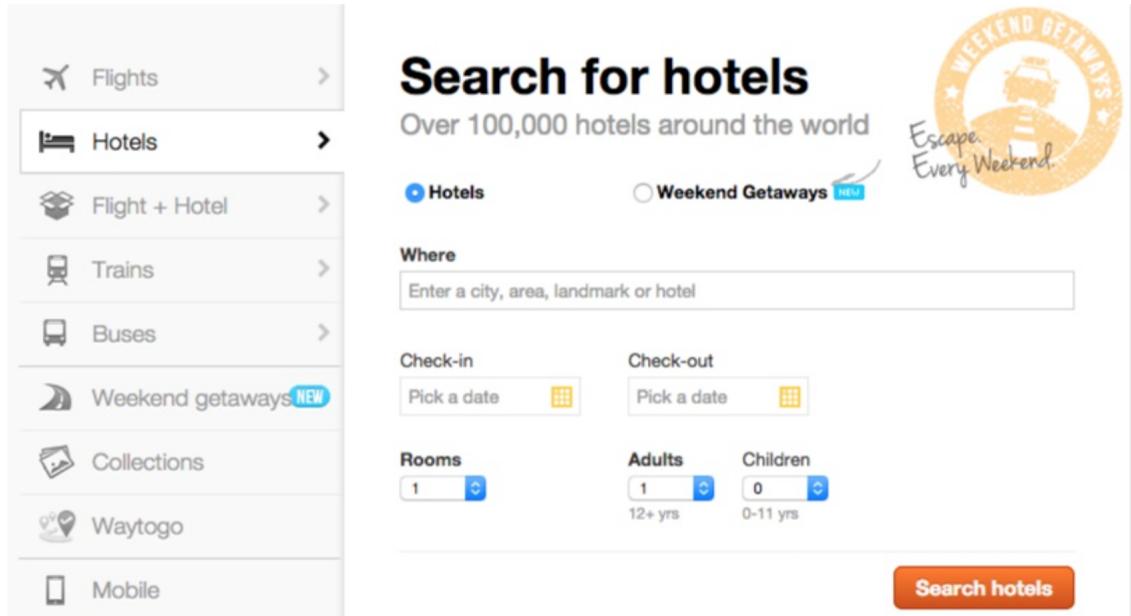


Figura 20 – Reconhecer é Melhor do Que Lembrar. Fonte: [Selvaraj]

imagem, já na segunda, existe um link em azul para opções mais avançadas, caso o usuário deseje outras opções além de simplesmente compartilhar

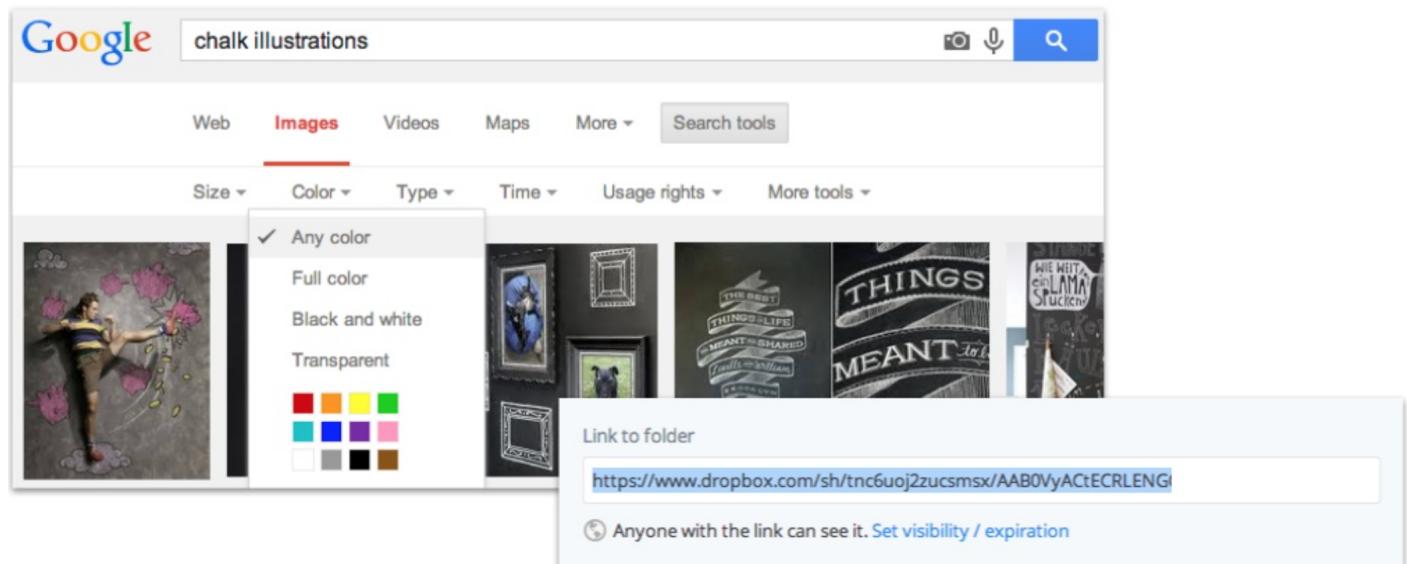


Figura 21 – Uso de "Aceleradores" Para Usuários Mais Experientes. Fonte: [Selvaraj]

8. **Estética e *Design Minimalista*:** os diálogos não precisam conter informações que não são utilizadas com frequência. Toda informação extra compete com uma informação relevante o que diminui a visibilidade do que realmente é importante.

Ajude Usuários a Reconhecer, Diagnosticar e Se Recuperarem de Erros: Mensagens de erro, devem ser expressas da maneira mais clara possível, indicando o precisa-

mente qual o problema, e indicando uma possível solução. Na Figura 22 as duas imagens apresentam exemplos de como lidar com o erro. Na primeira imagem existe um destaque para as possíveis soluções, já na segunda, não são apresentadas soluções, mas sim possíveis formas de se contornar o problema.

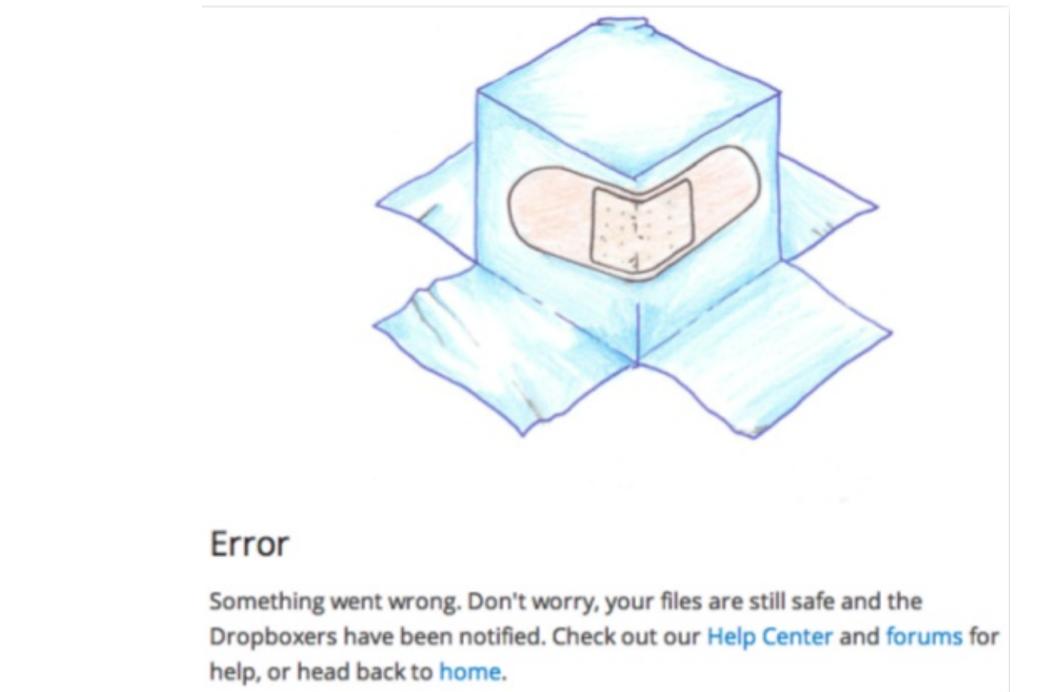


Figura 22 – Exemplo Mostrando Telas de Erro de Maneira a Tranquilizar o Usuário.
Fonte: [Selvaraj]

9. **Ajuda e Documentação:** Mesmo que a solução seja auto-explicativa, é necessário que exista uma documentação. Toda informação deve ser fácil de ser encontrada, e a lista de etapas para se concluir uma pesquisa à documentação não deve ser muito grande. Na primeira imagem da Figura 23, tem-se um manual simplificado que utiliza imagens para demonstrar um procedimento, ao invés de um longo manual. Na segunda imagem o uso de um ícone "?" apresenta uma breve descrição sobre as opções que estão disponíveis para o usuário.

2.6 Resumo do Capítulo

Para que se possa terceirizar 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 manutenibilidade.

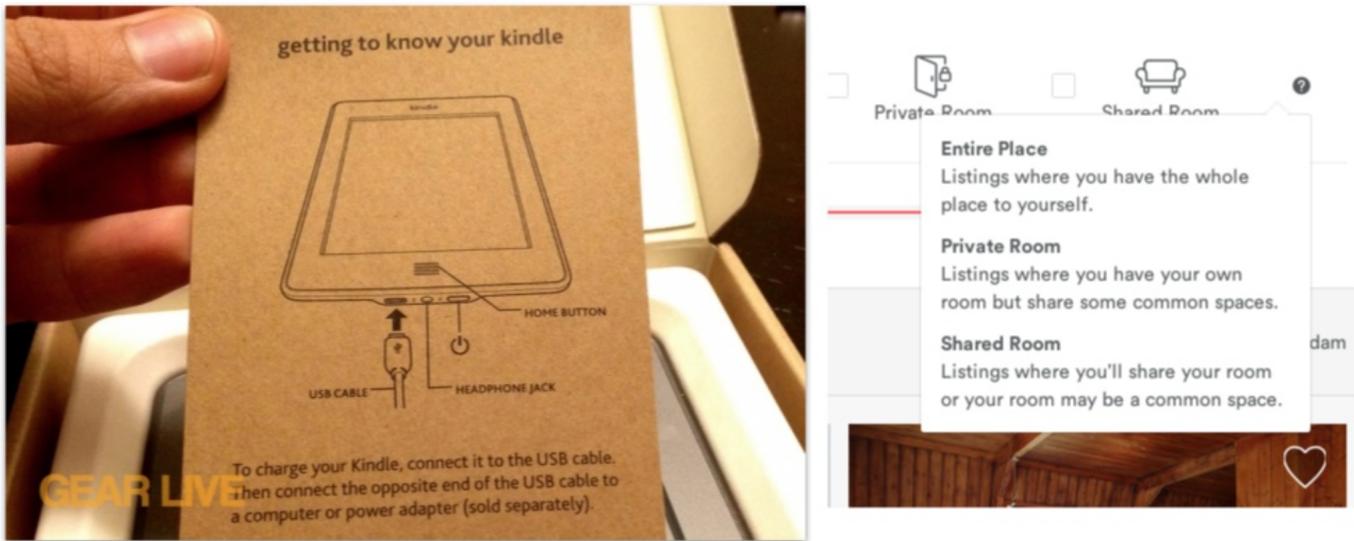


Figura 23 – Uso de Documentação Mesmo Para Pequenas Tarefas. Fonte: [Selvaraj]

Essa categoria é fundamental na contratação de software, pois permite saber o quanto 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 terceirizada 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. São apresentados dez heurísticas propostas por Nielsen, que servem como princípios fundamentais na criação de uma solução.

3 Suporte Tecnológico

O objetivo desta seção é descrever as principais tecnologias que nortearam e nortearão o desenvolvimento deste trabalho. Esta seção está dividida em Ferramentas para Programação e Ferramentas de Gerenciamento. Vale ressaltar que as ferramentas SonarQube, Google Charts e Charts.jsl orientam a presente proposta. As demais ferramentas poderiam ser facilmente substituídas por similares.

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 se deseja disponibilizar a solução futuramente para consulta e aprimoramento por parte dos interessados.

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 24. 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* que estendem as funcionalidades da ferramenta [Ferenc et al. 2014]. A arquitetura do SonarQube é composta

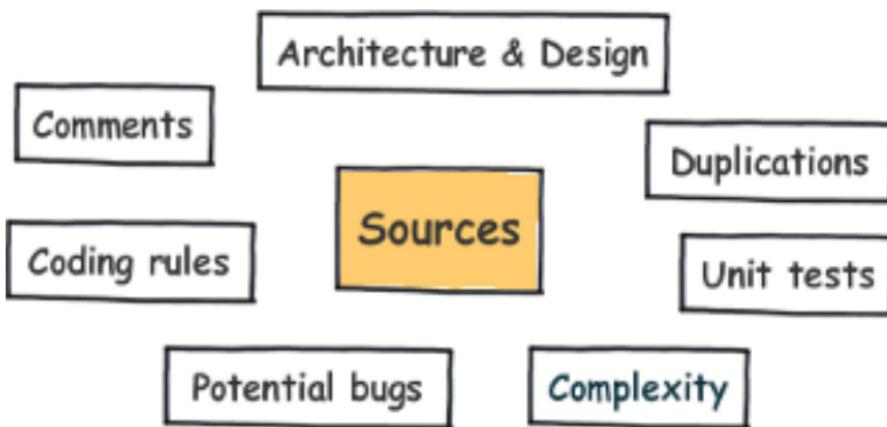


Figura 24 – Sete aspectos de qualidade de código cobertos pelo SonarQube. Fonte: [SonarQube documentation]

de quatro componentes, como pode ser visto na Figura 25 [SonarQube documentation].

1. Um **SonarQube Server** composto de três atividade principais:
 - a) Um **Web Server** para os desenvolvedores, gerentes que procuram *snapshots* da qualidade do código, e que pode ser usado para 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 bem como por 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.

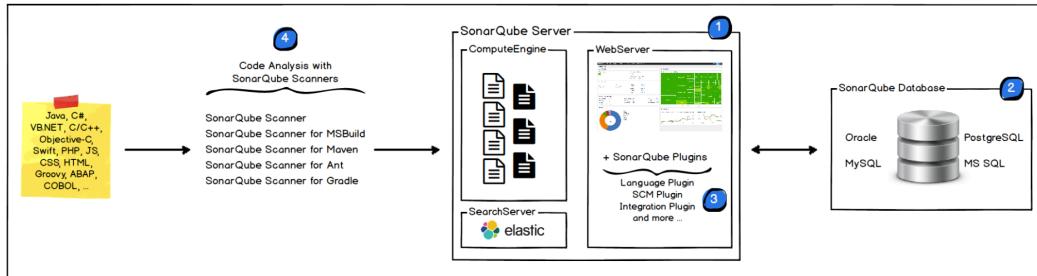


Figura 25 – Arquitetura SonarQube composta de quatro componentes: [SonarQube documentation]

O SonarQube também é facilmente integrado com outras ferramentas utilizadas durante o ciclo de vida do software. O esquema da Figura 26 apresenta a implantação do SonarQube em diversos estágios do desenvolvimento de software [SonarQube documentation].

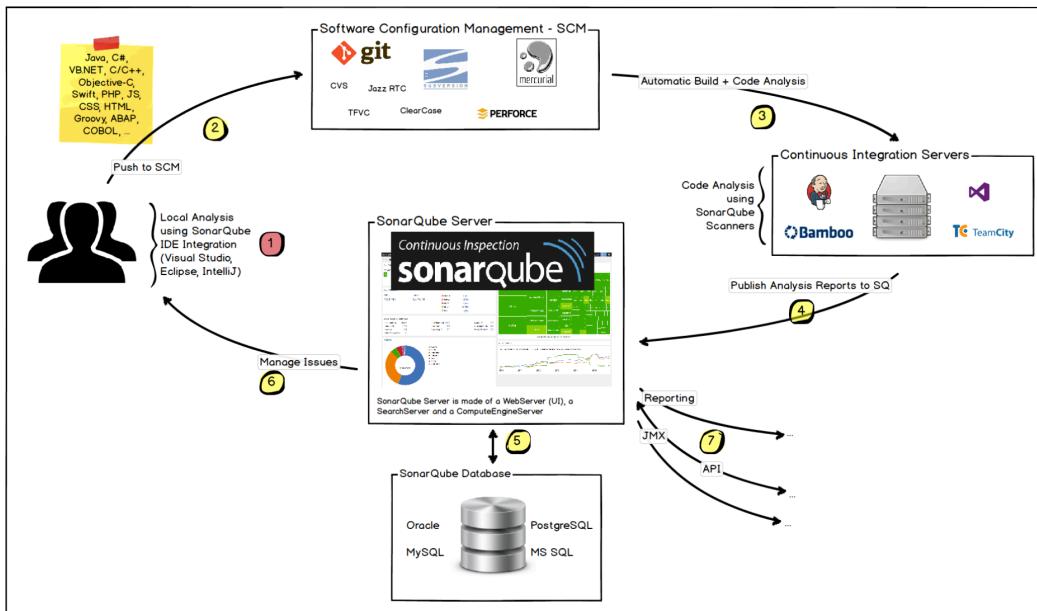


Figura 26 – Integração SonarQube em Diversas Áreas de Desenvolvimento de Software: [SonarQube documentation]

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 *Software Configuration Management* (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 dúvida técnica através do SonarQube UI.
7. Gestores recebem um relatório da análise.

A escolha do SonarQube como ferramenta de análise estática deu-se pelo fato de que grande parte dos Órgãos Públicos utiliza esta ferramenta para fazer a avaliação da qualidade dos produtos de software entregues pelas terceirizadas. Muito dos editais encontrados neste trabalho ([?], [?], [Mineral 2015]) utilizavam o SonarQube como uma das ferramentas de audição dos produtos de software entregues.

3.1.4 Google Charts e Charts.js

O *Google Charts* é uma API disponibilizada pelo Google. Fornece uma maneira para visualizar dados em um site. De gráficos de linha simples à 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 através de um JavaScript dentro do código HTML. Os gráficos são visualizados utilizando classes do JavaScript, e são renderizados através de HTML5/SVG, garantindo o funcionamento em diversos *browsers*.

Outra API utilizada é a Charts.js. Trata-se de 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 alguns gráficos que serão utilizados na criação do *dashboard* [[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 revisão bibliográfica.

3.2.1 Bonita

A ferramenta Bonita³ foi escolhida graças a sua facilidade de utilização e portabilidade para o sistema operacional Mac OS X bem como ao fato de ser gratuita. Esta ferramenta auxilia na modelagem de processos.

3.2.2 Mac OS X

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

3.2.3 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, referências, gráficos entre outros.

3.2.4 Sublime Text 3

O Sublime Text 3⁵ é um editor de texto bastante utilizado por programadores, por conferir 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 utilizado em muitos Órgãos do Governo por possuir código-aberto e uma grande comunidade trabalhando em sua evolução o software. Outras ferramentas importantes são o Google Charts e o Charts.js os quais são responsáveis por criar os gráficos que serão utilizados no *dashbord*. As demais ferramentas funcionam muito mais como ferramentas de apoio do que como ferramentas indispensáveis para a construção

³ <http://www.bonitasoft.com/>

⁴ <https://www.latex-project.org/>

⁵ <https://www.sublimetext.com/3>

⁶ <https://www.zotero.org>

da solução. Portanto, optou-se pela colocação dessas ferramentas nesse capítulo apenas para fins de documentação do material utilizado na solução.

4 Proposta

O objetivo deste capítulo é apresentar uma possível proposta de solução, a qual deve atender às necessidades colocadas no primeiro capítulo referente à problemática deste trabalho. A apresentação da proposta está concentrada em duas partes. A primeira parte está relacionada à coleta das métricas utilizando a ferramenta SonarQube, e a segunda parte está relacionada à criação do *dashboard* e à visualização das informações. Para análise da presente proposta, foram 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 os resultados obtidos através de uma avaliação qualitativa com possíveis usuários.

4.1 Ambiente Simulado

Para se simular um ambiente de produção, será utilizado como modelo de referência o ambiente apresentado por Luiza e Yago [Schaidt e Regis 2014]. No trabalho descrito, os autores caracterizam o órgão pertencente à APF como Órgão X. Uma das características referentes ao Órgão X que podem ser citadas diz respeito a sua área de jurisdição que abrange serviços de radiodifusão, postais e de telecomunicações. O Órgão X atualmente implanta o GeDDAS (Gestão de Demandas de Desenvolvimento Ágil de Software) proposto por Souza [Sobrinho 2014]. A Figura 27 apresenta o processo de desenvolvimento adotado pelo Órgão X.

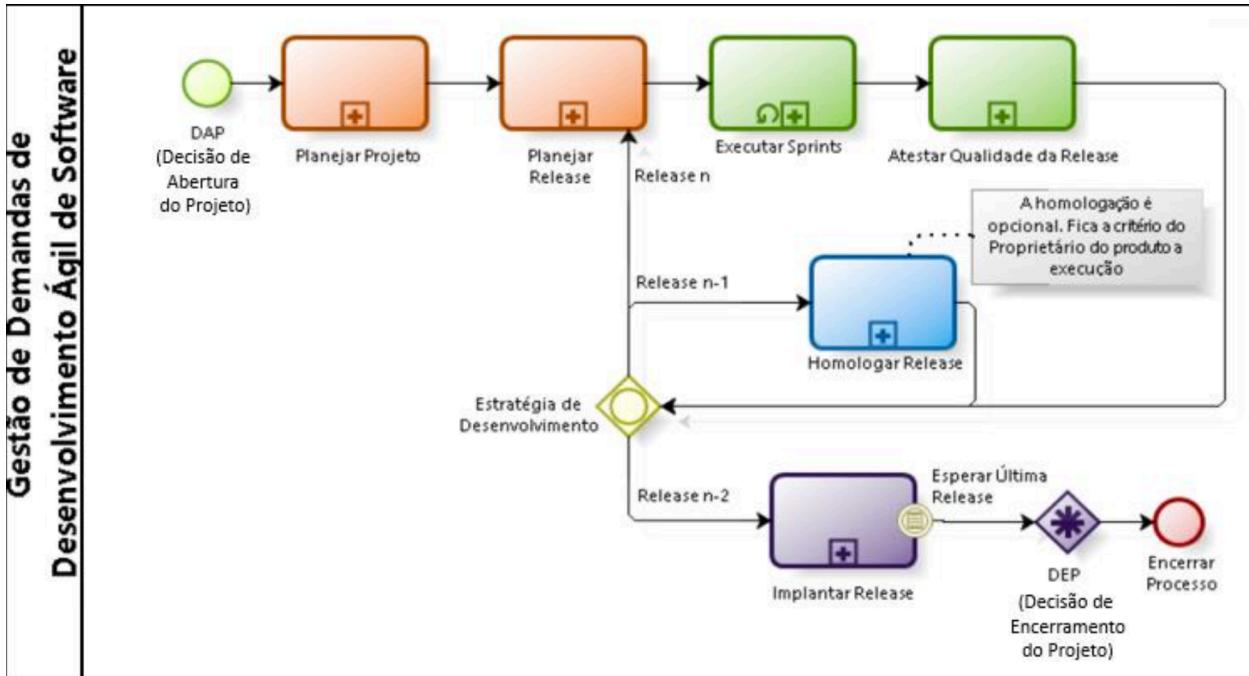


Figura 27 – Processo de Desenvolvimento Adotado Pelo Órgão X. Fonte: [Schaidt e Regis 2014]

A solução proposta tem o objetivo de atuar tanto do lado do Órgão Público, onde o Gestor acompanha o desenvolvimento do projeto, quanto da empresa contratada em que a ferramenta é integrada ào processo de desenvolvimento do software. A Figura 28 apresenta um diagrama que demonstra o ciclo de verificação juntamente com a solução apresentada. Na Figura é possível ver que o time de desenvolvimento da empresa contratada desenvolve o código que é submetido para uma análise onde é feita a verificação das métricas que foram estipuladas. Após essa análise o resultado é submetido para o *dashboard* onde o Gestor de TI e a equipe de qualidade do Órgão X podem acompanhar o projeto.

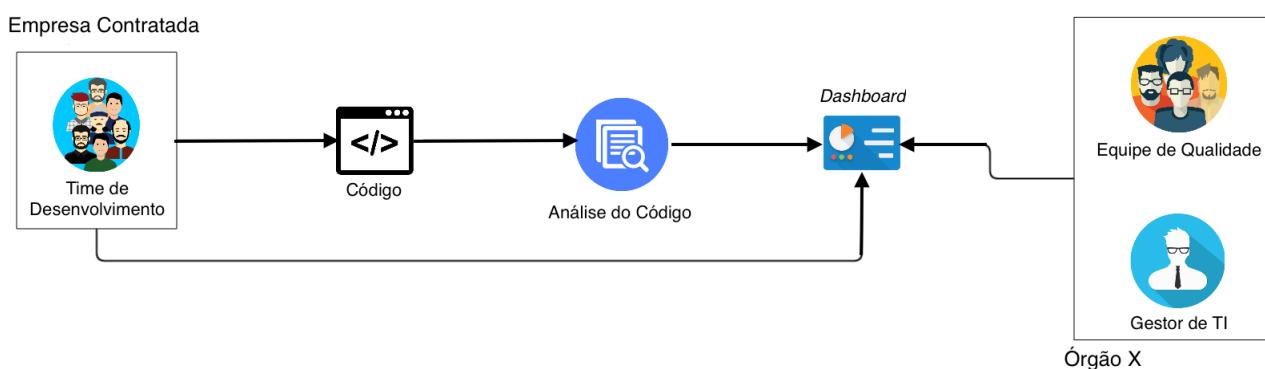


Figura 28 – Ciclo de Verificação Utilizando a Solução Compartilhada Entre Órgão Contratante e Empresa Contratada

4.2 Coleta das Métricas

Para fazer a coleta das métricas, será utilizada a ferramenta SonarQube. Como já informado na capítulo de Suporte Tecnológico, esse suporte é muito utilizado em Órgãos Públicos e em editais por ser uma ferramenta *open-source*. Nesse contexto, a coleta das métricas é feita de maneira dinâmica, onde o usuário decide qual suíte de métricas vai utilizar, podendo utilizar uma customizada ou utilizar uma como base. A presente proposta irá utilizar as suítes de métricas estabelecidas no edital do DNPM [[Mineral 2015](#)]). Portanto, serão utilizadas as suítes 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. Será possível ainda, 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 produtos de software entregues por empresas terceirizadas. Entretanto percebe-se que ainda que o trabalho possua grande aprovação, dificilmente o mesmo irá substituir o fator humano da auditoria. Portanto, o software compreende apenas um suporte que visa facilitar a avaliação geral do software entregue sob o ponto de vista da qualidade de código. Recomenda-se, uma vez que o software seja submetido à ferramenta e seja aceito, a realização de uma auditoria em cima de uma amostragem do software entregue, sob o olhar de um analista especializado para tal atividade.

4.3 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 dos projetos, com indicadores quanto à aprovação ou reprovação de cada projeto seguindo os limites estabelecidos pelo edital [[Mineral 2015](#)]. 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.

Para melhor compreensão da proposta, foram elaborados protótipos das possíveis telas que farão parte da solução final. A Figura 29 apresenta a tela inicial da solução, por onde o Gestor fará o acesso colocando seu *username* e seu *password*. Ainda nesta tela,

existe um *link* para caso o Gestor não se lembre do seu *password*, onde o sistema enviará uma mensagem, para o *email* cadastrado, solicitando a alteração.



Figura 29 – Tela Login

A página de inicial do projeto, também chamada de *home page*, aqui apresentada na Figura 30. Nesta página, o Gestor tem uma visão de todos os projetos, que ele cadastrou para serem acompanhados. Cada projeto é representado por um retângulo de uma cor , e neste retângulo estão algumas informações referentes ao projeto, que podem ser visualizadas, sem a necessidade de se abrir o projeto. Tanto as cores dos retângulos, como as informações dos projetos podem ser customizadas de acordo com o Gestor. Nesta página, ainda existe um botão que direciona o usuário para a tela de "adicionar projetos". Esta tela é somente para o gestor.



Figura 30 – Tela Home

A Figura 31 apresenta um protótipo do *dashboard* que será implementado. Nesta página encontram-se as informações referentes ao projeto selecionado. Esta página apresenta em forma de gráficos, as métricas que foram selecionadas pelo Gestor para aquele projeto. Esta é a única página que pode ser acessada pela empresa terceirizada. Em cada métrica é possível passar a seta seletora por cima do ícone "?" para que se tenha uma breve explicação da métrica

Para se fazer a adição de um novo projeto, é necessário que o projeto a ser adicionado, esteja armazenado em um repositório que o Gestor tenha acesso de leitura. A Figura 32 representa um protótipo da página de adição de projetos. Nesta página são adicionados, o nome do projeto, uma descrição, a url do repositório em que se encontra o projeto e por último, as métricas que serão analisadas. Assim como na página do *dashboard*, cada métrica apresenta um ícone "?" que apresenta uma breve explicação de cada métrica.

4.4 Avaliação

A avaliação do *dashboard* será feita por parte de um gestor de tecnologia de um Órgão 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

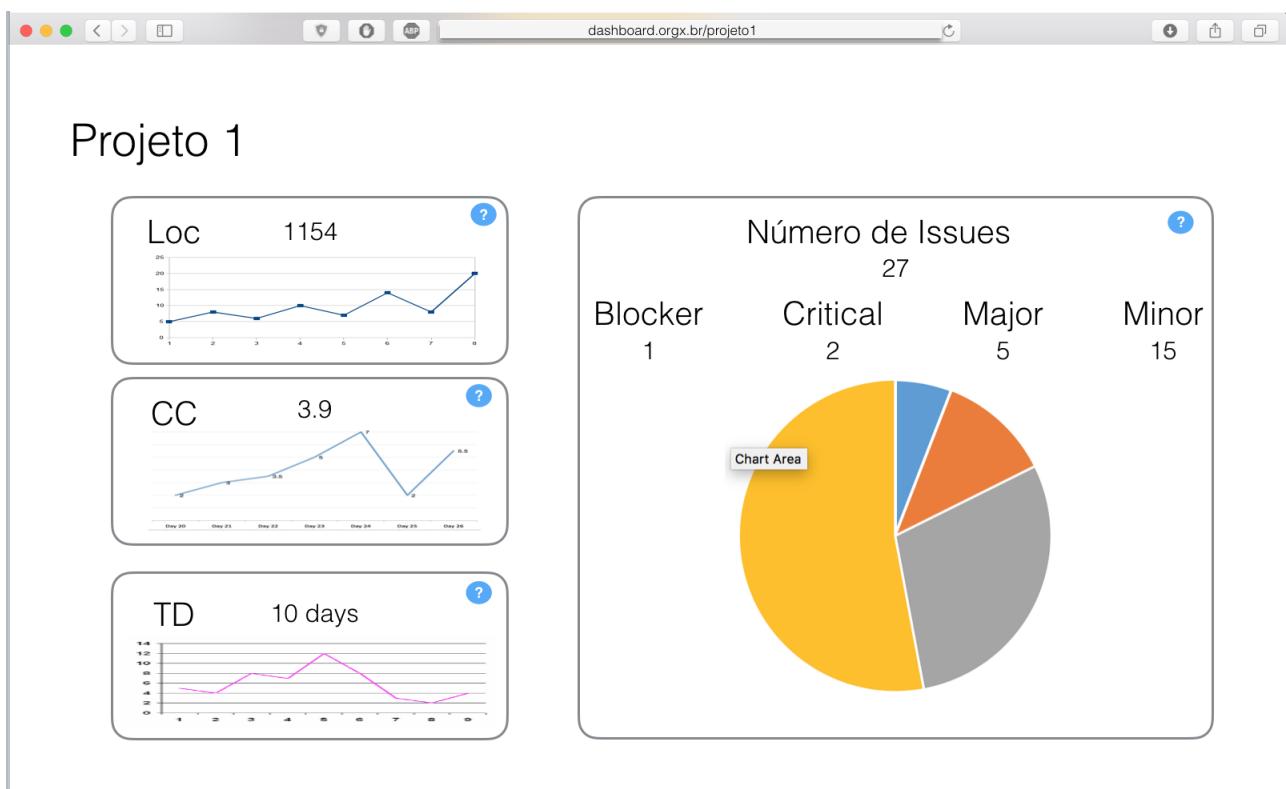


Figura 31 – Tela Dashboard

The screenshot shows a form for adding a new project:

- Nome do Projeto:** [Text input field]
- Descrição:** [Text area]
- url:** [Text input field]
- Métricas:** [Checkboxes]
 - LOC ?
 - TD ?
 - CC ?
 - Cobertura ?
 - Duplicação ?
 - Usar Edital ?
- Buttons:**
 - Salvar (Save) in blue button
 - Cancelar (Cancel) in red button

Figura 32 – Tela Adicionar

tal experiência com contratação de software para Órgãos Públícos, novamente avaliando aspectos como usabilidade e melhorias necessárias para implantação em um Órgão. Para fazer esta avaliação, o gestor responderá a um questionário contendo, inicialmente, cinco perguntas, referentes ao uso e às funcionalidades do software produzido. Uma primeira versão do questionário consta na Figura 33

4.5 Resumo do Capítulo

A proposta deste trabalho é criar uma maneira facilitada de acompanhar a qualidade de código estático dos produtos de software entregues pelas terceirizadas. Para fazer esta análise, a solução orienta-se por uma ferramenta de análise estática SonarQube e por um conjunto de métricas relacionadas às boas práticas de programação, algumas dessas métricas, foram retiradas dos livros Clean Code [Martin 2009] e Code Complete [McConnell 2004]. 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 qualidade do código. A última etapa deste trabalho consiste na avaliação do trabalho produzido, a qual será feita 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 33 – Questionário a ser aplicado ao fim do período de teste

5 Metodologia

Este capítulo tem como objetivo apresentar a metodologia utilizada para desenvolvimento do trabalho. A seção 5.1 apresenta a classificação da metodologia utilizada para o desenvolvimento do trabalho, juntamente com o plano metodológico. A seção 5.2 descreve o processo de desenvolvimento utilizado na criação da solução. Por último a seção 5.3 apresenta os cronogramas para condução do TCC 1, bem como do TCC 2.

5.1 Metodologia de Pesquisa

Segundo [Moresi 2003], a pesquisa pode ser entendida como um conjunto de ações que tem como objetivo encontrar um problema, sendo construída através de procedimentos empíricos e sistemáticos. Para Moresi, existem quatro classificações básicas para a pesquisa, quanto: à natureza, à abordagem, ao objetivo e ao meio pelo qual é feita a investigação. A Figura 34 apresenta em quais classificações esse trabalho se baseia.



Figura 34 – Seleção das Características Metodológicas. Fonte: [Moresi 2003]

Este trabalho tem um caráter mais voltado para pesquisa aplicada, por envolver características específicas na contratação de software do Governo Brasileiro. Outra característica que determina este trabalho como pesquisa aplicada é a natureza do trabalho ser voltada para o uso nas áreas de TI dentro dos Órgãos Públicos.

Segundo Tatiana e Denise [Gerhardt e Silveira 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. Segundo as autoras [Gerhardt e Silveira 2009], outra característica inerente a este tipo de pesquisa é

a observação do mundo social ao mundo natural. Portanto, tal característica apresenta-se de maneira muito forte quando, ao propor a solução, procura-se adotar um conjunto de métricas que são utilizadas no mercado ao invés de outros conjuntos apresentados por outros autores. Oposto à pesquisa qualitativa, os resultados obtidos podem ser quantificados. A pesquisa quantitativa teve como fundamento o pensamento positivista lógico, prima pelas regras da lógica e do raciocínio dedutivo [Gerhardt e Silveira 2009].

Para Gil [Gil 2002], a pesquisa descritiva é focada em analisar características de uma população, fenômeno ou a relação entre as variáveis que as compõem. Este tipo de pesquisa não visa explicar os fenômenos que estão descrevendo [Moresi 2003]. Este trabalho apresenta o caráter descritivo ao tratar da natureza das contratações de software para Órgão Públicos Brasileiros, ou quando se fala de um processo de manutenção de software.

Outra característica deste trabalho é a escolha por fazer uma pesquisa de laboratório. Moresi destaca que a pesquisa de laboratório atua em um ambiente controlável, quando o pesquisador não tem a possibilidade de atuar em campo. Neste trabalho, a pesquisa em campo se torna algo complicado, pois é difícil conseguir acesso aos 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 recriadas as mesmas condições de uma situação em campo, porém, com o controle de um ambiente simulado. Outro meio de pesquisa é a pesquisa bibliográfica. A pesquisa bibliográfica é feita através do aprofundamento de materiais já publicados [Gerhardt e Silveira 2009]. Este trabalho também utilizou pesquisa bibliográfica durante a fase de Iniciação, em que o objetivo era estudar propostas similares à definida neste trabalho.

5.1.1 Plano Metodológico

O plano metodológico consiste em duas fases: Iniciação e Execução. Durante o TCC1 será implementada somente a primeira fase. A fase de Execução será implementada no TCC2. A primeira fase (Figura 35) possui seis atividades principais, são elas: Definir Tema, Validar o Escopo, Elaborar um Roteiro de Pesquisa, Pesquisar Referências, Refinar Pesquisa e Catalogar Material Encontrado.

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 o tema, o escopo é validado juntamente com os professores orientadores para que haja um acordo entre aluno e orientador sobre o que será trabalhado.

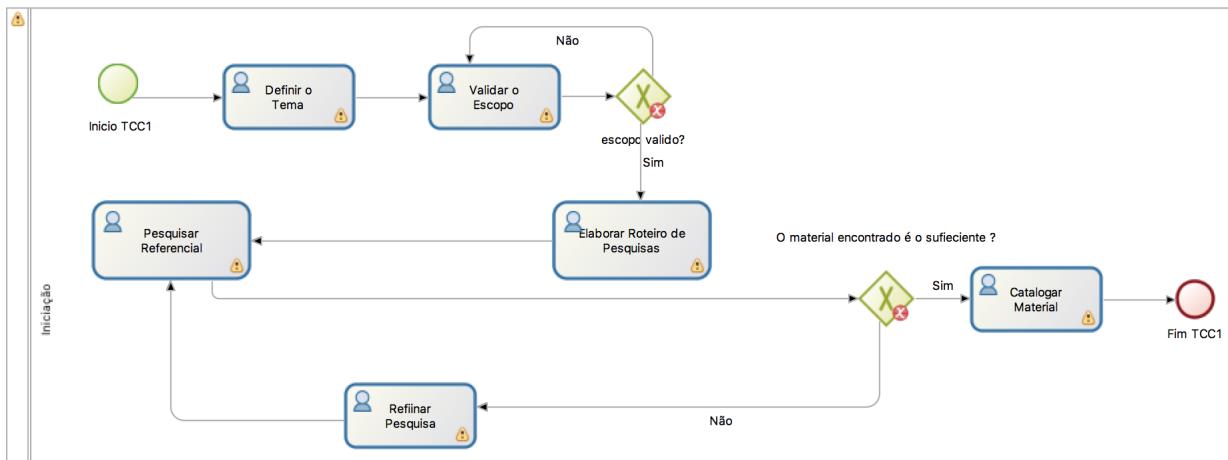


Figura 35 – Modelagem da Fase de Iniciação

Posteriormente, é necessário Elaborar um Roteiro de Pesquisa que consiste em encontrar a melhor *string*, trabalhando a revisão bibliográfica. Na presente proposta, as *strings* foram montadas de acordo com o tema da pesquisa, portanto, não houve uma string geral que fosse utilizada por todo o processo investigativo. Dessa forma, as primeiras strings foram montadas com base em conceitos chave nos tópicos e áreas de interesse para a presente proposta. Algumas das strings utilizadas inicialmente foram "contratação de software", "criação de um *dashboard*" e "métricas de qualidade". Com o decorrer do trabalho, no processo investigativo, a proposta foi adquirindo um escopo cada vez mais refinado, específico e conciso, culminando na seguinte string de busca final: "utilização de *dashboard* + acompanhamento de métricas + manutenibilidade + Órgão Públicos".

As atividades 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 não atendesse à temática do trabalho, 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 os principais pontos do material. 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 conforme ilustra a Figura 36.

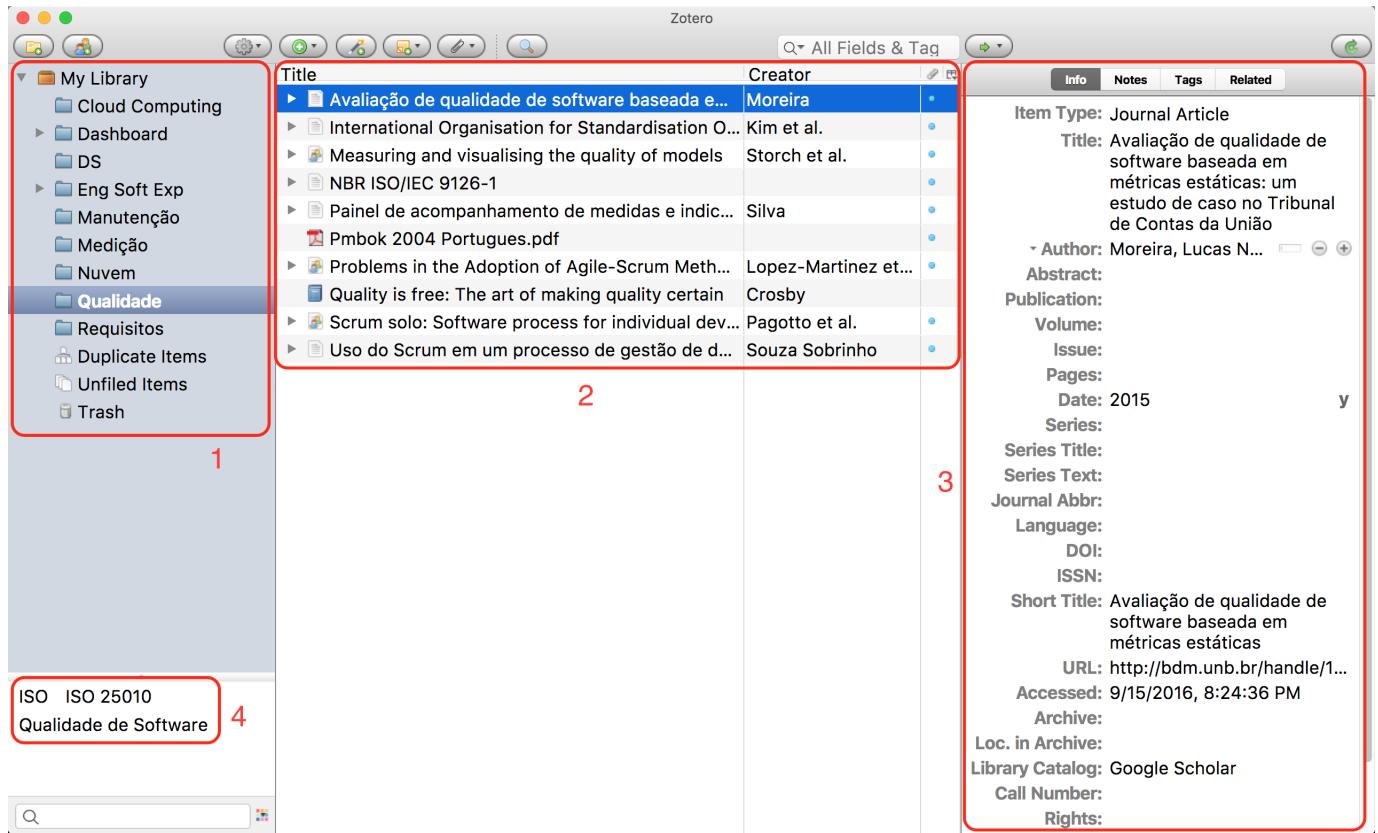


Figura 36 – Screenshot do Zotero contendo as categorias dos materiais pesquisados, suas tags e anotações

- 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, o mesmo era exportado para o Bibdesk 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 Execução.

A Fase de Execução (Figura 37) 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

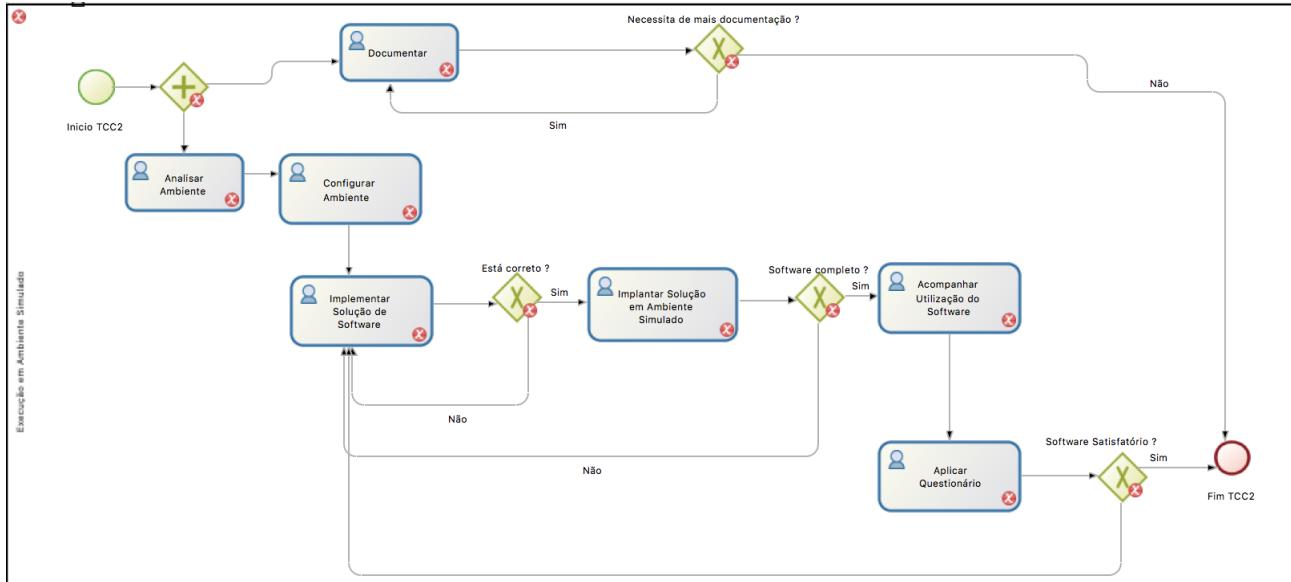


Figura 37 – Modelagem da Fase de Execução

as melhores ferramentas e a 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 orientando-se pelas configurações de um ambiente real.

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 ocorra um implemento funcional de software. Ao fim de cada iteração, se o software produzido estiver funcionando e testado, ele é colocado no ambiente simulado, equivalente a um ambiente de produção, para que se possa acompanhar o seu funcionamento.

Uma vez que a solução esteja pronta para uso, esta será disponibilizada para um conjunto de utilizadores que irão testar a solução bem como documentar as suas impressões quanto à 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 quão pertinente e adequada foi a solução.

Com a Fase de Execução finalizada, encerra-se o processo. Nesta segunda fase, os principais artefatos são: a solução funcionando e um refinamento do documento entregue na primeira fase. A Figura 38 apresenta uma visão geral do processo que será aplicado neste trabalho.

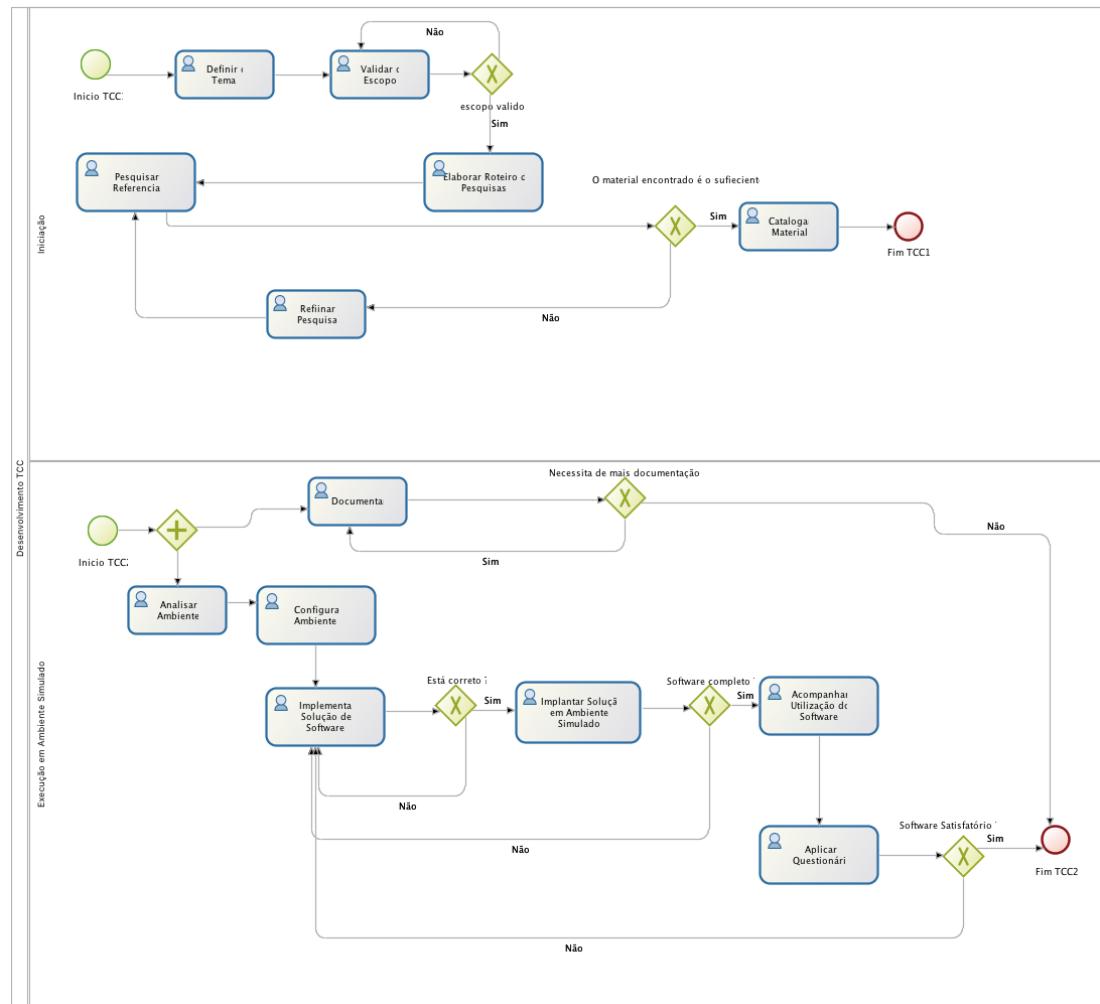


Figura 38 – Plano de Pesquisa

5.2 Metodologia de Desenvolvimento

O *dashboard* será criado utilizando uma adaptação da metodologia de desenvolvimento Scrum [Pagotto et al. 2016]. O Scrum é uma metodologia de desenvolvimento de software baseada em princípios de desenvolvimento ágil. As metodologias ágeis ganharam popularidade por serem adaptáveis a times pequenos, ou projetos que possuam prazos curtos na entrega do software, e ainda projetos com requisitos constantemente alterados [Lopez-Martinez et al. 2016]. Como o Scrum é um modelo de desenvolvimento iterativo e incremental, propõe que o projeto de desenvolvimento seja estruturado em pequenas entregas, chamadas de *sprints*. Essas *sprints* podem ser associadas a pequenos ciclos de desenvolvimento, os quais 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 39 pode-se observar que 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].

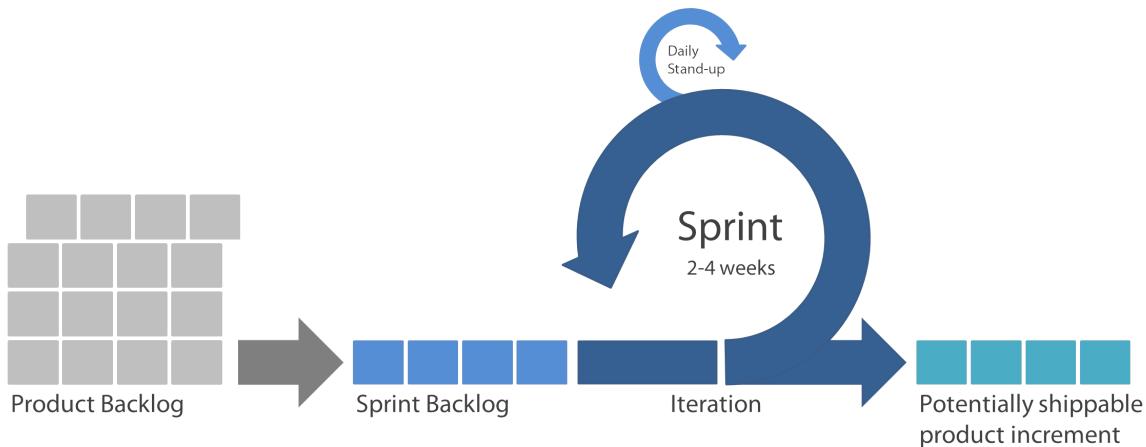


Figura 39 – Ciclo de Desenvolvimento do Scrum

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].
- **User Story:** 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 40. A confirmação ou critérios de aceitação é um *checklist* abordando o que deve ser verificado para que a história seja dada como concluída.

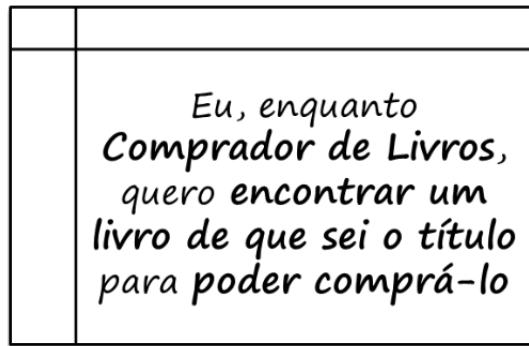


Figura 40 – Exemplo de História de Usuário. Fonte: [Sabbagh 2014]

- **Story Point:** Unidade relativa que caracteriza o esforço da equipe de desenvolvimento para finalizar uma atividade. A escala utilizada é definida pela equipe de desenvolvimento. Neste trabalho, será utilizado a escala de Fibonacci (1,2,3,5,8,13, ...).
- **Kanban:** Forma de visualização de atividades muito utilizadas com cartões, os quais são movimentados em um quadro determinando o status da atividade, conforme mostra a Figura 41.

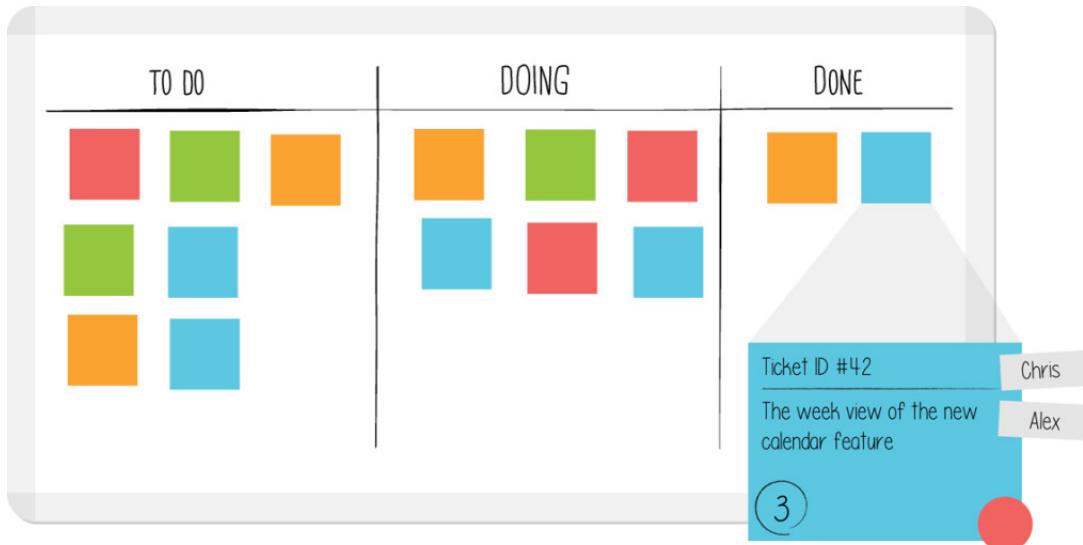


Figura 41 – Exemplo de *Kanban*

Por se tratar de uma adaptação do Scrum, algumas modificações foram necessárias para que a metodologia possa atender às necessidades do trabalho. Uma das alterações que foram feitas é quanto ao time de desenvolvimento. Para este projeto o time de desenvolvimento será de uma pessoa, diferente do Scrum tradicional, que define a quantidade de membros em um time de desenvolvimento deve ser entre 3 a 9 pessoas [Sabbagh 2014]. Outra adaptação, é quanto ao *Product Owner*(P.O) que no Scrum tradicional tem o papel

de garantir o retorno do investimento para os *stakeholders*. Nesse trabalho o papel de P.O será realizado pelo Orientador deste trabalho. No papel de cliente, será utilizado os próprios usuários que realizarão a avaliação da solução.

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 poderá sofrer alterações. O cronograma foi dividido em duas tabelas (Tabela 4 e Tabela 5) referentes às atividades do TCC1 e TCC2 respectivamente.

5.3.1 Cronograma TCC1

A Tabela 4 apresenta um cronograma em relação às 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 às 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 projeto, a natureza da pesquisa pode ser classificada como Aplicada, devido ao uso específico nas áreas de TI, para uma situação específica que é a contratação de software. Quanto à abordagem, a pesquisa é Hibrida, pois existe momentos em que a pesquisa assume um caráter Qualitativo (forma de se analisar e coletar os dados é feita de maneira empírica), contudo, existe um lado Quantitativo na pesquisa (ao que se refere à pesquisa que é feita com o possíveis usuários). Quanto ao objetivo, essa pesquisa tem um caráter Descritivo, pois 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 é Laboratorial e Bibliográfico. Laboratorial pois, todo o desenvolvimento da pesquisa é feita em um ambiente controlado e não em campo, e Bibliográfico, pois foi feito um levantamento bibliográfico durante a primeira fase do projeto.

Quanto à metodologia de desenvolvimento, optou-se por uma adaptação da metodologia Scrum. Tal escolha deu-se pelo fato de que o framework da metodologia é adaptável para times de desenvolvimento pequenos e com entregas de produtos de software em curtos períodos de tempo.

6 Resultados Parciais

Neste capítulo, estão descritos os resultados que foram obtidos até o momento. No caso, tem-se 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 a elaborações e o desenvolvimento desse projeto, as primeiras atitudes tomadas foram aprofundar mais o conhecimento no assunto e garantir que será possível importar as métricas extraídas 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 as mesmas 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 encontra-se 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 serão levantadas 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 de 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 49.
- BRASIL. *Lei n 8.666, de 21 de junho de 1993*. 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 52.
- 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.
- CROSBY, P. B. *Quality is free: The art of making quality certain*. [S.l.]: Signet, 1980. Citado na página 22.
- DOUMONT, J.-L.; Vandebroeck, Philippe. 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 50.
- FEW, S. *Information Dashboard Design - The effective Visual Comunication of Data*. [S.l.]: O'Reilly, 2006. Citado 6 vezes nas páginas 13, 38, 40, 41, 42 y 43.
- FILHO, C. F. *História da computação: O Caminho do Pensamento e da Tecnologia*. [S.l.]: EDIPUCRS, 2007. Citado na página 21.
- GERHARDT, T. E.; Silveira , D. T. *Métodos de pesquisa*. [S.l.]: Universidade Aberta do Brasil, 2009. Citado 2 vezes nas páginas 63 y 64.
- GIL, A. C. *Como elaborar projetos de pesquisa*. São Paulo (SP): Atlas, 2002. OCLC: 817765297. ISBN 978-85-224-3169-4. Citado na página 64.
- 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 52.

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.l.]. Citado 3 vezes nas páginas 13, 31 y 42.

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

KUSTERS, R. J.; HEEMSTRA, F. J. Software maintenance: an approach towards control. In: *Software Maintenance, 2001. Proceedings. IEEE International Conference on*. [S.l.: s.n.], 2001. p. 667–670. ISSN 1063-6773. Citado na página 32.

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 68.

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 Elektrotehnika*, v. 111, n. 5, p. 123–128, 2011. Disponível em: <<http://ecomam.ktu.lt/index.php/elt/article/view/372>>. Citado na página 69.

MARTIN, R. C. *Clean code: a handbook of agile software craftsmanship*. [S.l.]: Pearson Education, 2009. Citado na página 61.

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 49.

MCCONNELL, S. *Code complete*. [S.l.]: Pearson Education, 2004. Citado na página 61.

- 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, 52 y 57.
- 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. 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 14, 63 y 64.
- 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.
- NIELSEN, J. *10 Usability Heuristics for User Interface Design*. Citado na página 42.
- 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 68.
- 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%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%A3o+%C3%A9+necess%C3%A1rio+ter+uma+ampla+compreens%C3%A3o+do+Scrum+para%22+&ots=ERSywNIEAd&sig=rvmJD_BXcQY-P0MYZBqQJOaOcOA>. Citado 4 vezes nas páginas 14, 68, 69 y 70.
- 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, L.; Regis, Y. *Monitoramento de Qualidade de Software na Administração Pública Federal*. Dissertação (Mestrado) — Universidade de Brasília, 2014. Citado 5 vezes nas páginas 14, 22, 29, 55 y 56.

SCHNAIDER, L. Uma abordagem para Medição e Análise em Projetos de Desenvolvimento de Software. *Brasília, SBQS*, 2004. Citado na página 22.

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.

SELVARAJ, S. *Ten Usability Heuristics With Example*. Citado 7 vezes nas páginas 13, 43, 44, 45, 46, 47 y 48.

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.

SOBRINHO, L. P. d. S. Uso do Scrum em um processo de gestão de demandas de desenvolvimento de software por terceiros para um órgão público federal brasileiro. 2014. Disponível em: <<http://bdm.unb.br/handle/10483/8216>>. Citado na página 55.

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 14, 50 y 51.

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