

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-

83 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

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, 07 de dezembro 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



# Agradecimentos

Agradeço primeiramente a Deus, tenho 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 abriram 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 em alguns momentos, palavras 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 fase 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, de uma adaptação da metodologia Scrum para o desenvolvimento da solução, contudo algumas mudanças foram necessárias, principalmente devido ao escopo e ao número de papéis, que a metodologia exige. 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 Departments. 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 Departments. For this solution, the dashboard works as a tool for visualizing indicators, as determined by the Public Department Project Manager. The dashboard displays the project indicators, which are being developed by the Employer, for the Manager. The work consisted of two phases: Initiation and Execution. The objective of the first phase is to establish fundamentals (theoretical and architectural) for the Execution phase, while the focus of the second phase lies in the implementation of the solution. The use of an adaptation of the Scrum methodology for the development of the solution was chosen, but some changes were necessary, mainly due to the scope and number of roles required by the methodology. 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.



# Listas de ilustrações

Figura 1 – Relação entre as NBR ISO/IEC 9126 e NBR ISO/IEC 14598 .Fonte: [NBR ISO/IEC 9126-1 2016] . . . . .	26
Figura 2 – Modelo de Qualidade para Qualidade Interna e Externa .Fonte: [NBR ISO/IEC 9126-1 2016] . . . . .	27
Figura 3 – Produto de Qualidade de Software.Fonte: [ISO 25010] . . . . .	29
Figura 4 – Diagrama das Atividades de Manutenção de Software.Fonte: [Pfleeger e Bohner 1990] . . . . .	31
Figura 5 – SR por Colaboração. Adaptado de: [Sarwar et al. 2001] . . . . .	35
Figura 6 – SR por Conteúdo. Adaptado de: [Recommender systems-How they works and their impacts 2006] . . . . .	36
Figura 7 – Tipos de Visualização Mais Frequentes. Fonte: [Schwendimann 2016] . . . . .	37
Figura 8 – Exemplo de Gráfico de Barra Utilizando Google Charts . . . . .	37
Figura 9 – Exemplo de Gráfico de Linhas utilizando HiCharts . . . . .	38
Figura 10 – Exemplo de Gráfico de Pizza utilizando Chart.js . . . . .	38
Figura 11 – Exemplo de <i>Network Graph</i> utilizando Google Charts . . . . .	38
Figura 12 – Exemplo 1 de <i>Dashboard</i> apresentado por Stephen Few. Fonte: [Few 2006] . . . . .	40
Figura 13 – Exemplo 2 de <i>Dashboard</i> apresentado por Stephen Few. Fonte: [Few 2006] . . . . .	41
Figura 14 – Exemplo 3 de <i>Dashboard</i> apresentado por Stephen Few. Fonte: [Few 2006] . . . . .	42
Figura 15 – Sete aspectos de qualidade de código cobertos pelo SonarQube. Fonte: [SonarQube documentation] . . . . .	48
Figura 16 – Arquitetura SonarQube composta de quatro componentes: [SonarQube documentation] . . . . .	49
Figura 17 – Integração SonarQube em Diversas Áreas de Desenvolvimento de Software: [SonarQube documentation] . . . . .	49
Figura 18 – Processo de Desenvolvimento Adotado Pelo Órgão X. Fonte: [Schaidt e Regis 2014] . . . . .	54
Figura 19 – Ciclo de Verificação Utilizando a Solução Compartilhada Entre Órgão Contratante e Empresa Contratada . . . . .	54
Figura 20 – Diagrama de Implantação da Solução no Órgão X . . . . .	55
Figura 21 – Exemplo de Perfis Coletados Após Questionário . . . . .	56
Figura 22 – Métricas Extraídas do SonarQube . . . . .	57
Figura 23 – Tela Login . . . . .	58
Figura 24 – Tela Home . . . . .	58

Figura 25 – Tela Dashboard . . . . .	59
Figura 26 – Tela Adicionar . . . . .	60
Figura 27 – Página do <i>Dashboard</i> da Prova de Conceito . . . . .	60
Figura 28 – Seleção das Características Metodológicas. Fonte: [Moresi 2003] . . . . .	63
Figura 29 – Modelagem da Fase de Iniciação . . . . .	65
Figura 30 – <i>Screenshot</i> do Zotero contendo as categorias dos materiais pesquisados, suas <i>tags</i> e anotações . . . . .	66
Figura 31 – Modelagem da Fase de Execução . . . . .	67
Figura 32 – Plano de Pesquisa . . . . .	68
Figura 33 – Ciclo de Desenvolvimento do Scrum . . . . .	69
Figura 34 – Exemplo de História de Usuário. Fonte: [Sabbagh 2014] . . . . .	70
Figura 35 – Exemplo de <i>Kanban</i> . . . . .	70
Figura 36 – Formulário de Consentimento . . . . .	83

# **Lista de tabelas**

Tabela 1 – Índice de Cobertura Por Tipo de Teste do Edital da DNPM. Fonte: [Mineral 2015] . . . . .	25
Tabela 2 – Métricas de Qualidade de Código Exigidas pela DNPM. Fonte: [Mineral 2015] . . . . .	25
Tabela 3 – Mapeamento de Cores e seus Significados . . . . .	39
Tabela 4 – Principais Características de Quatro Questionários de Avaliação. Fonte: Adaptado de [Sauro e Lewis 2016] . . . . .	43
Tabela 5 – Cronograma TCC1 . . . . .	71
Tabela 6 – Cronograma TCC2 . . . . .	71
Tabela 7 – <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>
SR	Sistema de Recomendação
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

<b>1</b>	<b>INTRODUÇÃO</b>	<b>19</b>
1.1	Contextualização	19
1.2	Problema de pesquisa	20
1.3	Justificativa	20
1.4	Objetivos	21
1.4.1	Objetivos Gerais	21
1.4.2	Objetivos Específicos	21
1.5	Resultados Esperados	22
1.6	Organização do trabalho	22
<b>2</b>	<b>REFERENCIAL TEÓRICO</b>	<b>23</b>
2.1	Processo de Contratação de Software na APF	23
2.1.1	Avaliação da Qualidade Em Um Processo de Contratação	25
2.2	<b>Qualidade</b>	<b>26</b>
2.2.1	Norma SQuaRE	27
2.2.2	Manutenção de Software	29
2.3	<b>Métricas de Qualidade de Software</b>	<b>31</b>
2.3.1	Suíte de Chidamber-Kemerer	31
2.3.2	Suíte MOOD	33
2.4	<b>Aprendizado de Máquina e Sistemas de Recomendação</b>	<b>34</b>
2.5	<b>Visualização da Informação</b>	<b>36</b>
2.5.1	Dashboard	37
2.6	<b>Questionarios de Avaliação</b>	<b>42</b>
2.7	<b>Personas</b>	<b>43</b>
2.8	<b>Resumo do Capítulo</b>	<b>44</b>
<b>3</b>	<b>SUPORTE TECNOLÓGICO</b>	<b>47</b>
3.1	<b>Ferramentas de Programação</b>	<b>47</b>
3.1.1	GIT	47
3.1.2	Github	47
3.1.3	SonarQube	48
3.1.4	Google Charts e Charts.js	50
3.2	<b>Ferramentas de Gerenciamento</b>	<b>50</b>
3.2.1	Bonita	51
3.2.2	Mac OS X	51
3.2.3	LaTeX	51

3.2.4	Sublime Text 3 . . . . .	51
3.2.5	Zotero . . . . .	51
<b>3.3</b>	<b>Resumo do Capítulo . . . . .</b>	<b>51</b>
<b>4</b>	<b>PROPOSTA . . . . .</b>	<b>53</b>
<b>4.1</b>	<b>Ambiente Simulado . . . . .</b>	<b>53</b>
<b>4.2</b>	<b>Coleta das Métricas . . . . .</b>	<b>53</b>
<b>4.3</b>	<b>Criação do <i>Dashboard</i> . . . . .</b>	<b>57</b>
<b>4.4</b>	<b>Avaliação . . . . .</b>	<b>60</b>
<b>4.5</b>	<b>Resumo do Capítulo . . . . .</b>	<b>61</b>
<b>5</b>	<b>METODOLOGIA . . . . .</b>	<b>63</b>
<b>5.1</b>	<b>Metodologia de Pesquisa . . . . .</b>	<b>63</b>
5.1.1	Plano Metodológico . . . . .	64
<b>5.2</b>	<b>Metodologia de Desenvolvimento . . . . .</b>	<b>68</b>
<b>5.3</b>	<b>Cronograma . . . . .</b>	<b>71</b>
5.3.1	Cronograma TCC1 . . . . .	71
5.3.2	Cronograma TCC2 . . . . .	71
<b>5.4</b>	<b>Resumo do Capítulo . . . . .</b>	<b>72</b>
<b>6</b>	<b>RESULTADOS PARCIAIS . . . . .</b>	<b>73</b>
6.0.1	<i>Status</i> Atual do Projeto . . . . .	73
6.0.2	Próximas Atividades . . . . .	73
	<b>REFERÊNCIAS . . . . .</b>	<b>75</b>
	<b>APÊNDICES</b>	<b>81</b>
	<b>APÊNDICE A – FORMULÁRIO DE CONSENTIMENTO . . . . .</b>	<b>83</b>

# 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 a solução proposta. Os resultados dessa pesquisa são apresentados na seção Resultados Esperados. Por fim, tem-se a Organização do Trabalho, a qual procura apresentar os principais capítulos desta 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 2.271 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 na produção de software é uma área muito ampla, e que abrange desde qualidade da arquitetura de software, a 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 pelas terceirizadas, 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 em desenvolvimento. 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

A motivação deste trabalho ocorreu durante um dos projetos em que participei na faculdade. O projeto se tratava de uma parceria entre a academia e um órgão público federal. Neste projeto existiam diversas áreas de trabalho, sendo uma delas Qualidade de Software, a qual tive a oportunidade de trabalhar, juntamente com outros dois alunos, Luiza Schaidt e Yago Regis.

Pelo fato de as atividades desenvolvidas no projeto afetarem diretamente algumas terceirizadas que tinham contrato juntamente com o órgão, se fazia constante reuniões com as três partes: órgão, academia e terceirizada. Com o decorrer do projeto, fui ganhando maturidade para começar a discernir algumas das práticas adotadas pela terceirizada.

Uma das práticas que mais me chamava atenção era o fato de que o produto de software entregue pela terceirizada, era um produto de qualidade ruim, que na maioria das vezes dava defeito quando já estava em produção. Comecei a perceber que essa prática ocorria por falta de um controle rigoroso do órgão que na ilusão de ter um software funcionando acabava aceitando o produto entregue ignorando o que estava por trás da funcionalidade entregue.

Desde então comecei a perceber que o trabalho de implantar um ambiente que incentivasse a produção com qualidade de software seria ineficaz enquanto não houvesse um acompanhamento por parte do órgão em verificar a qualidade do software entregue durante o processo de desenvolvimento pela terceirizada.

## 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, a partir 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 e 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 se detalha 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 mantenabilidade 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 n 2.271 de 1997 [BRASIL 1997] dispõe sobre a contratação de serviços pela APF. Segundo o Decreto, todos os produtos ou serviços que não apresentam relação direta com o propósito da Instituição do Governo Federal devem ser 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 n 8.666 de 1993, o Governo Brasileiro para garantir a isonomia (princípio geral do direito segundo o qual todos são iguais perante a lei), diz que para contratações de bens ou serviços deve-se dar prioridade para licitação [BRASIL 1993]. A licitação pode ocorrer de acordo com quatro categorias: concorrência, tomada de preços, convite e pregão [BRASIL 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 implementadas 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 terceirizada 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. 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 (DNPM) [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]

<b>Tipo de Teste</b>	<b>% de cobertura</b>
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]

<b>Métrica</b>	<b>Meta</b>	<b>Severidade</b>
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. O que tem se vivenciado na realidade brasileira na área de computação é que o software entregue pelas empresas brasileiras, em sua maioria, é um software com uma qualidade ruim. 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 especificações 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]

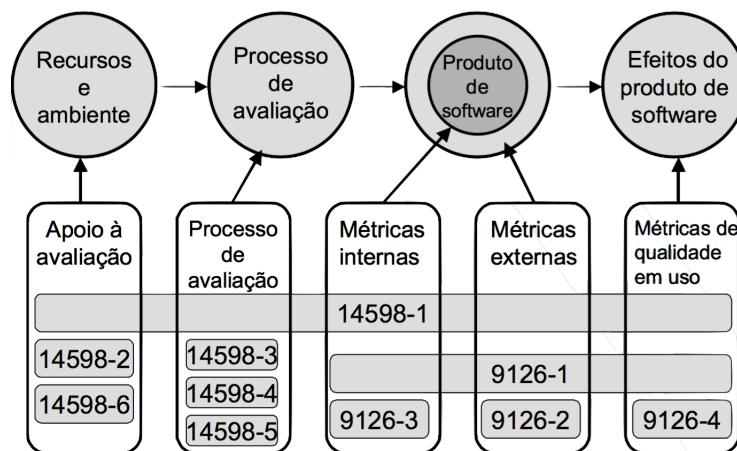


Figura 1 – Relação entre as NBR ISO/IEC 9126 e NBR ISO/IEC 14598 .Fonte: [NBR ISO/IEC 9126-1 2016]

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 sub características, medidas por meio de métricas internas e externas.

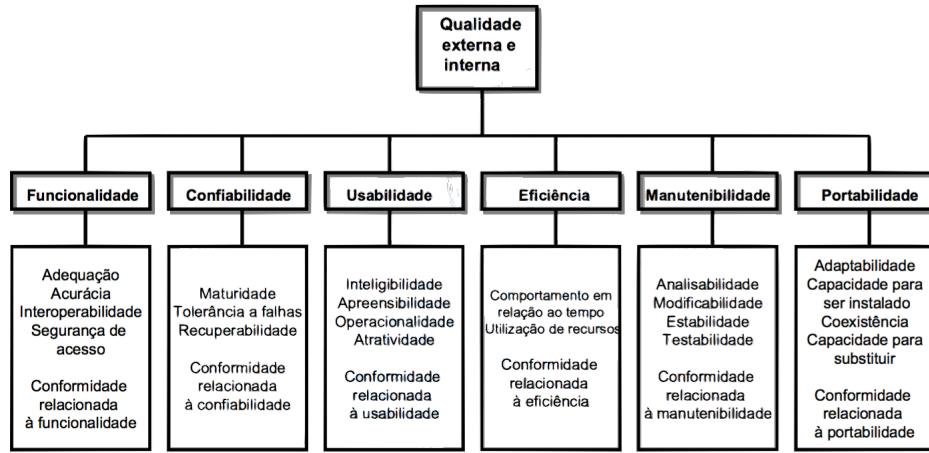


Figura 2 – Modelo de Qualidade para Qualidade Interna e Externa .Fonte: [NBR ISO/IEC 9126-1 2016]

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

- **Funcionalidade:** Capacidade do produto de software de prover funções que atendam às necessidades explícitas e implícitas, quando o software estiver sendo utilizado sob condições especificadas.
- **Confiabilidade:** Capacidade do produto de software de manter um nível de desempenho especificado, quando usado em condições especificadas.
- **Usabilidade:** Capacidade do produto de software de ser compreendido, aprendido, operado e atraente ao usuário, quando usado sob condições especificadas.
- **Eficiência:** Capacidade do produto de software de apresentar desempenho apropriado, relativo à quantidade de recursos usados, sob condições especificadas.
- **Manutenibilidade:** Capacidade do produto de software de ser modificado. As modificações podem incluir correções, melhorias ou adaptações do software devido a mudanças no ambiente e nos seus requisitos ou especificações funcionais.
- **Portabilidade:** Capacidade do produto de software de ser transferido de um ambiente para outro.

Em 2011, surgiu um conjunto de normas 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 estendidos 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.

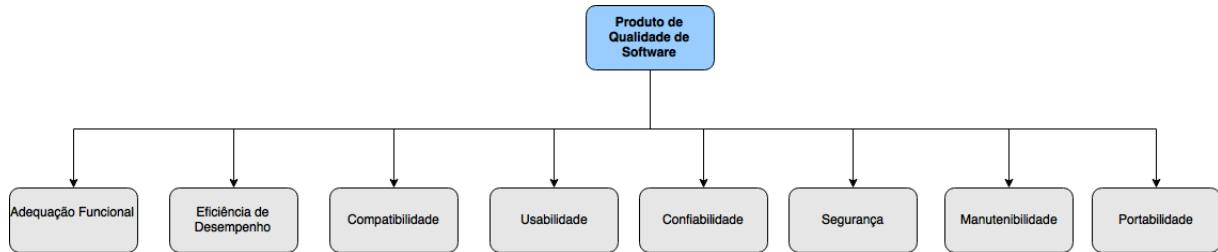


Figura 3 – Produto de Qualidade de Software. Fonte: [ISO 25010]

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, 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 subprocessos, 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 manutenibilidade 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 última 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

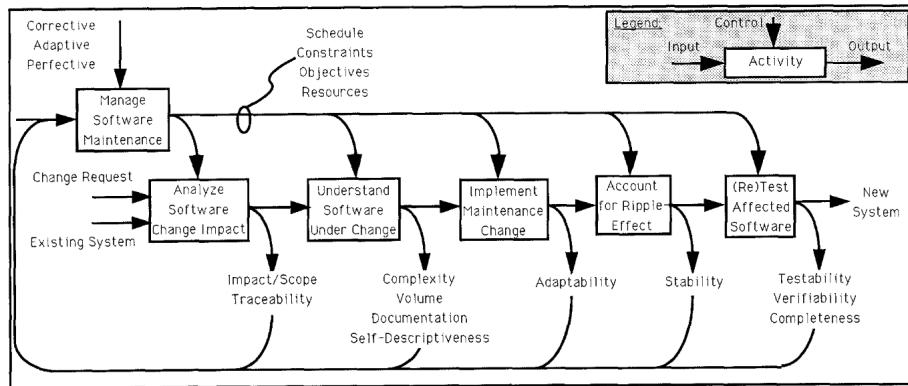


Figura 4 – Diagrama das Atividades de Manutenção de Software. Fonte: [Pfleeger e Bohner 1990]

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 está 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 Aprendizado de Máquina e Sistemas de Recomendação

Aprendizado de Máquina se caracteriza pela implementação de técnicas que ajudam a melhorar o desempenho de um software aprendendo através de conhecimento indutivo [Mitchell e Learning 1997]. Maria Carolina [Monard e Baranauskas 2003] diz "A indução é a forma de inferência lógica que permite obter conclusões genéricas sobre um conjunto particular de exemplos. Ela é caracterizada como o raciocínio que se origina em um conceito específico e o generaliza, ou seja, da parte para o todo. Na indução, um conceito é aprendido efetuando-se inferência indutiva sobre os exemplos apresentado". Esse conhecimento se baseia no conceito que modelos são obtidos através de um conjunto de dados ou representações de experiências [Peres et al. 2012]. Podem ser divididos em duas categorias: aprendizagem supervisionada e não supervisionada [Russell 2009].

- **Aprendizagem Supervisionada:** o algoritmo recebe dados exemplos de entradas e de saída e com base nesses exemplos define uma regra.
- **Aprendizagem Não Supervisionada:** o algoritmo recebe apenas dados de entrada e o próprio algoritmo define os conjuntos e as saídas desses conjuntos.

Para que seja possível o aprendizado de máquina são necessários algoritmos que irão conduzir esse aprendizado. Um dos algoritmos é o por análise de agrupamento *clustering*. Esse algoritmo serve para agrupar um conjunto de objetos de forma que os objetos que sejam mais semelhantes estejam no mesmo grupo (*cluster*). Algumas das formas de

se fazer esse agrupamento são utilizando a distância entre os pontos, aproximação por centróides entre outras.

Segundo Lucas [Brunialti et al. 2015] um Sistema de Recomendação (SR) tem por objetivo principal sugerir itens que possam satisfazer a necessidade de um usuário sob um determinado aspecto. No livro "*Recommender systems: an introduction*" [Jannach et al. 2010] são apresentados modelos de SR que se caracterizam pela forma como o SR é implementado. O primeiro modelo é o colaborativo, neste modelo as recomendações são feitas através de itens que os usuários interagiram no passado e relacionando essas informações. A figura 5 apresenta um exemplo de SR por recomendação em que o item "laranja" é recomendado para o usuário 3 pelo fato de que tanto o usuário 1 quanto ao usuário 3 compraram "maçã", nesta lógica se o usuário 1 comprou "laranja" o usuário 3 também pode querer maçã.

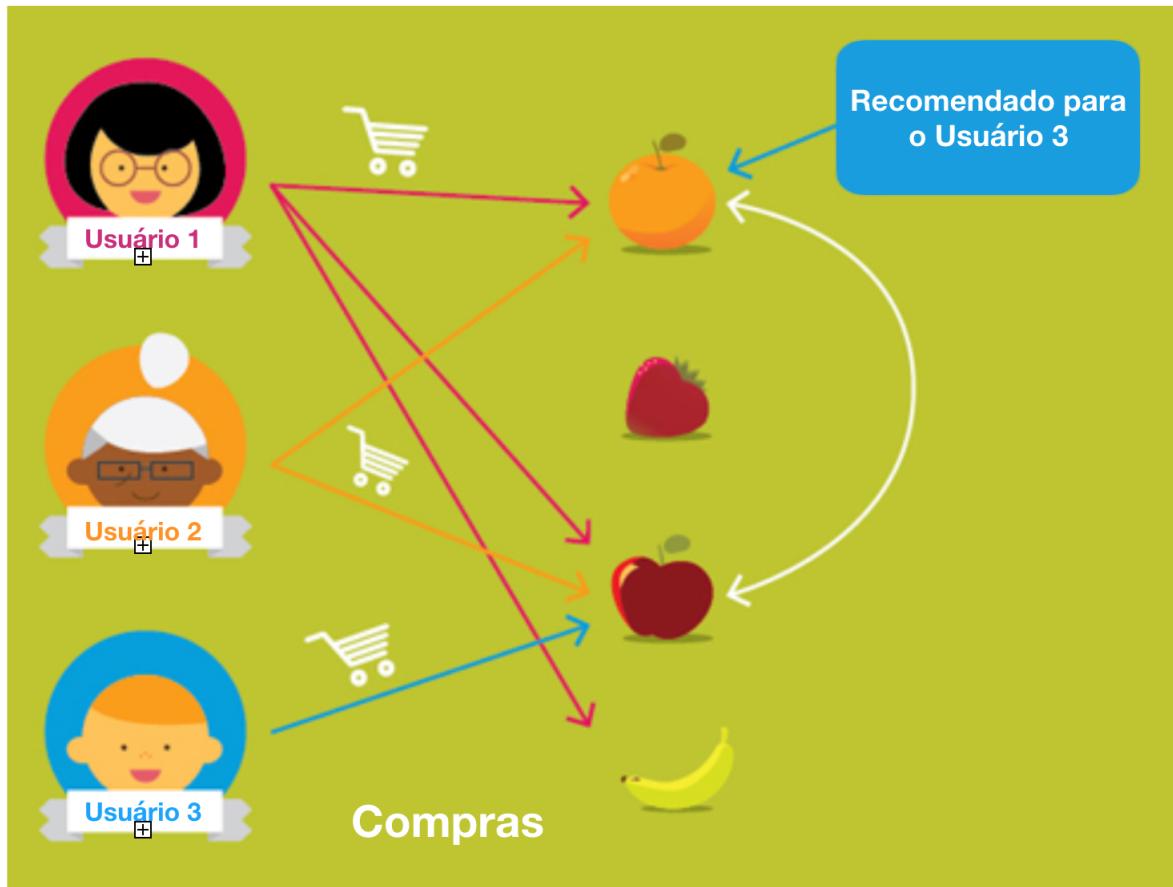


Figura 5 – SR por Colaboração. Adaptado de: [Sarwar et al. 2001]

O segundo modelo é o com base no conteúdo, onde as recomendações são obtidas através das características dos itens e no perfil do usuário. A figura 6 apresenta um modelo de SR com base em conteúdo, em que as avaliações feitas pelo usuário são comparadas com as avaliações feitas por outros usuários.

Existe ainda um terceiro modelo que se baseia no conhecimento em que as re-

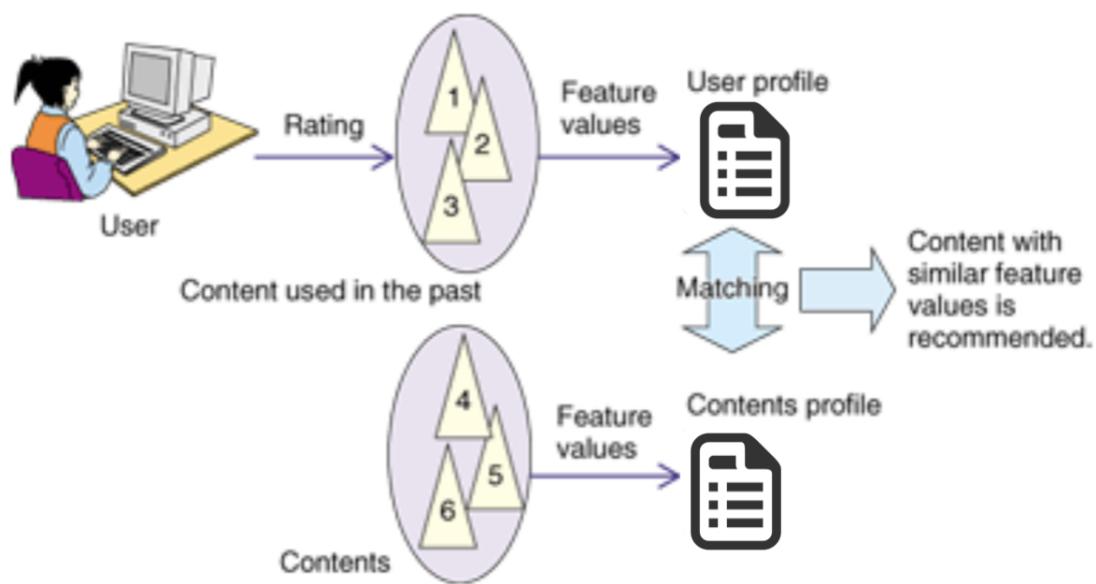


Figura 6 – SR por Conteúdo. Adaptado de: [Recommender systems-How they works and their impacts 2006]

comendações são feitas apartir de uma dedução da necessidade ou preferência de um usuário.

## 2.5 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 está imagem vai além de representações gráficas ou conceitos.

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.

### 2.5.1 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 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).

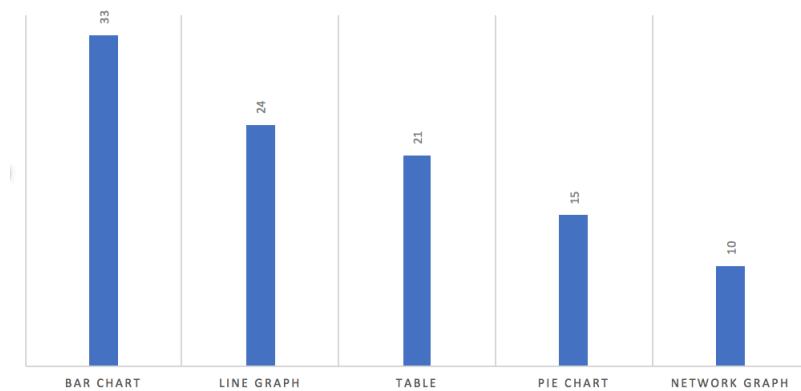


Figura 7 – Tipos de Visualização Mais Frequentes. Fonte: [Schwendimann 2016]

- 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]

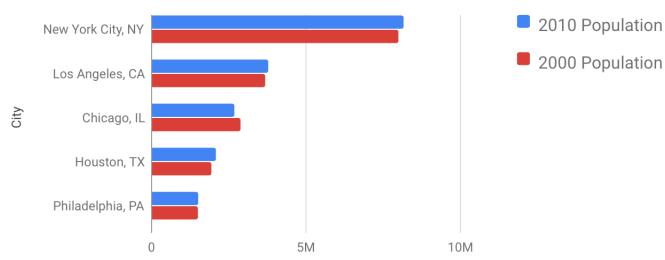


Figura 8 – Exemplo de Gráfico de Barra Utilizando Google Charts

- 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.

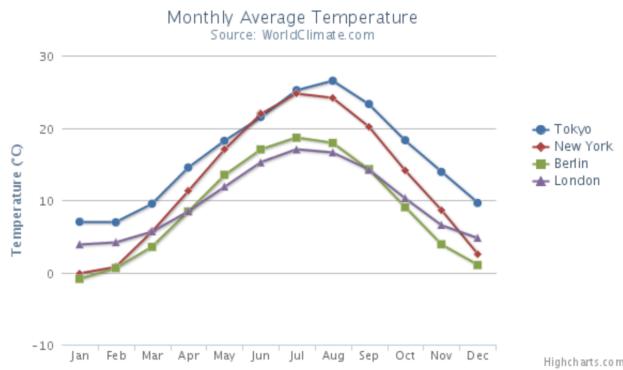


Figura 9 – Exemplo de Gráfico de Linhas utilizando HiCharts

- 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.

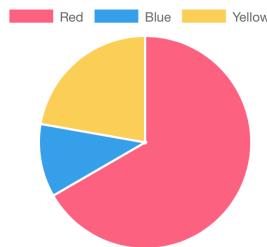


Figura 10 – Exemplo de Gráfico de Pizza utilizando Chart.js

- *Network Chart*: este tipo de visualização reforça relacionamentos entre entidades. As entidades são mostradas como nós e os relacionamentos como linhas (Figura 11).

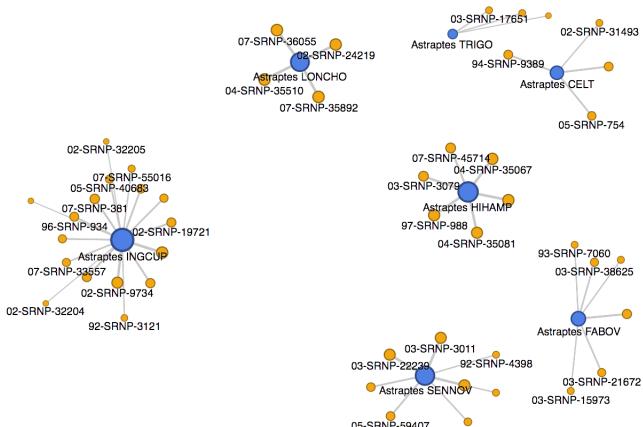


Figura 11 – Exemplo de *Network Graph* utilizando Google Charts

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:

"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.

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

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.



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

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* 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.

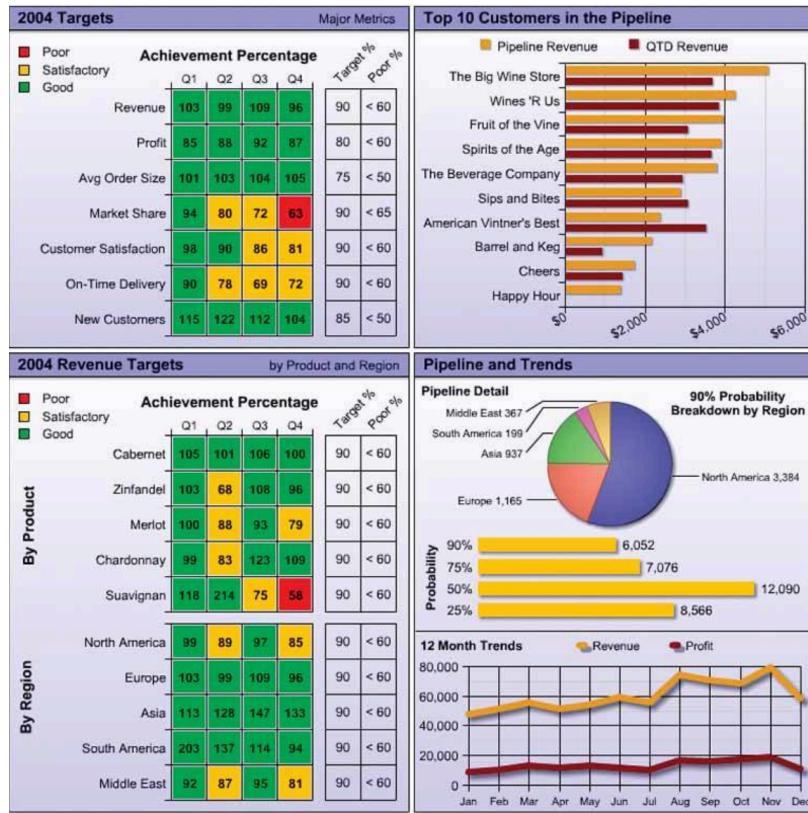


Figura 13 – Exemplo 2 de *Dashboard* apresentado por Stephen Few. Fonte: [Few 2006]

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

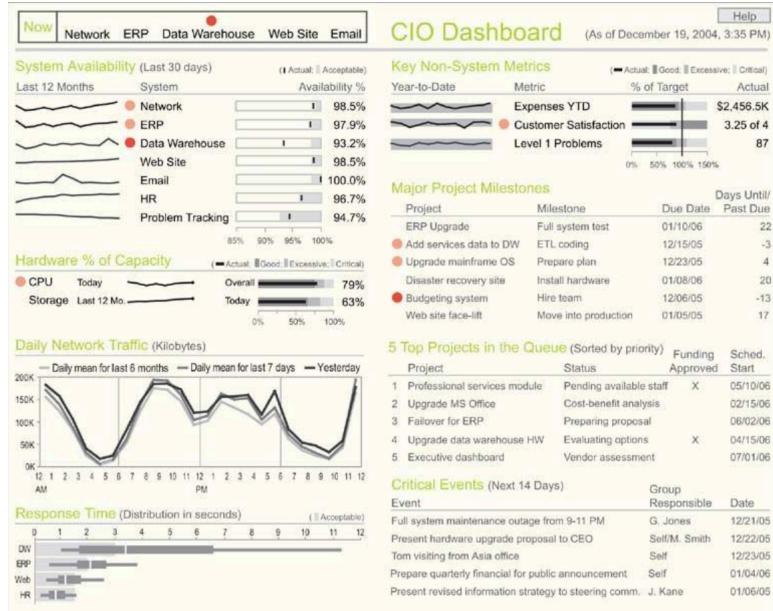


Figura 14 – Exemplo 3 de *Dashboard* apresentado por Stephen Few. Fonte: [Few 2006]

## 2.6 Questionários de Avaliação

Segundo Nunnally [Nunnally 1978], a padronização da medição de indicadores em questionários na área de usabilidade tem como principais vantagens:

- **Objetividade:** permite que diferentes pesquisadores tomem como base os mesmos indicadores.
- **Replicabilidade:** garante que seja fácil a replicação de estudos feitos por outros pesquisadores, ou até mesmo seus próprios estudos.
- **Quantificação:** medições padronizadas permitem que o pesquisador colete resultados com um grau de especificidade maior do que se fossem coletados baseados nos seus próprios julgamentos.
- **Economia:** desenvolver medições padronizadas requer um esforço muito grande. Entretanto uma vez que este já foi desenvolvido, o reuso se torna muito mais fácil.
- **Comunicação:** se torna fácil a comunicação entre pesquisadores quando se tem um processo de medição padronizado.
- **Generalização Científica:** é o coração do trabalho científico. Padronização é essencial para garantir a generalização dos resultados.

Sauro [Sauro e Lewis 2016] estabelece quatro questionários de avaliação de usabilidade. A tabela 4 apresenta um comparativo entre diferentes questionários. Dentre os

Tabela 4 – Principais Características de Quatro Questionários de Avaliação. Fonte: Adaptado de [Sauro e Lewis 2016]

Questionário	Valor da Licença	Número de Itens	Número de Subescalas	Confiabilidade Global
QUIS	\$50 - 750	27	5	0.94
SUMI	0 - 1000	50	5	0.92
PSSUQ	Gratuito	16	3	0.94
SUS	Gratuito	10	2	0.92

questionários abordados escolheu-se o questionário SUS (*Software Usability Scale*) para se realizar a avaliação do projeto. Dois fatores foram levados em consideração para a escolha do questionário, preço e número de perguntas.

Brooke [Brooke et al. 1996] define o questionário SUS como sendo uma escala de usabilidade "rápida e suja". O SUS se baseia em um questionário do tipo "*Likert Scale*", questionários deste tipo apresentam afirmações em que o entrevistado deve julgar o quanto concorda ou discorda da afirmação (no caso do questionário SUS, essa escala varia de 1 a 5).

A escala SU é geralmente utilizada depois que o entrevistado teve a oportunidade de usar o sistema que será avaliado. Deve ser instruído ao entrevistado que se responda imediatamente após ler a pergunta, não se deve pensar muito na resposta pois acaba induzindo o resultado. Todos os itens devem ser marcados e caso o entrevistado não saiba responder um item em particular deve-se marcar o ponto ao centro da escala.

Para calcular o resultado do questionário SUS deve-se somar os valores atribuídos em cada item, sendo que os itens 1,3,5,7 e 9 devem ser subtraídos 1 do valor aferido no questionário ( $X-1$ ), e nos itens 2,4,6,8 e 10 deve ser subtraído de 5 o valor aferido no questionário ( $5-X$ ). Feito a soma deve-se multiplicar o resultado obtido 2,5. A escala do Questionário SUS varia de 0 até 100.

## 2.7 Personas

O método Personas foi criado para ser utilizado no desenvolvimento de soluções de TI, contudo a técnica se tornou comum em outras áreas como desenvolvimento de produtos e na área de *marketing*. Personas são abstrações de um grupo de consumidores reais que dividem um conjunto de características e necessidades em comum [Pruitt e Adlin 2010]. Personas não podem ser confundidas com estereótipos de uma pessoa. O principal aspecto da descrição de uma persona é que não se deve olhar para toda as características da pessoa em destaque, mas somente para as características relevantes ao contexto.

Uma persona deve ser descrita em forma de narrativa. Segundo Cooper [Maness, Miaskiewicz e Sumner 2008] a descrição em forma de narrativa tem dois objetivos princi-

pais, fazer com que a persona se pareça com uma pessoa real e prover uma história que expresse as necessidades da persona no contexto do produto que está sendo criado. A narrativa de uma persona começa com a descrição do tipo de indivíduo que aquela persona representa, são detalhados, gostos, costumes, profissão, características físicas. Feito isso, são detalhadas as necessidades específicas daquela persona, são especificados quais são seus objetivos e metas relacionados com o contexto. Estas são as mesmas necessidades que seriam encontradas em um documento de requisitos, contudo estão escritas em forma de narrativa e associadas a uma persona específica [Manning, Temkin e Belanger 2003].

Supondo que se queira criar um site de compra de passagens, um exemplo de persona seria.

Bruce Wayne tem 50 anos, é casado com Diana Prince. Bruce é um bibliotecário que trabalha na Biblioteca de Gotham City. Uma vez por ano Bruce tira férias de 30 dias para viajar com Diana. Todas as vezes que Bruce viaja ele compra as passagens e reserva o hotel em agências de viagem especializadas. Bruce tem preferência por viajar para lugares com clima frio e que não sejam muito procurados por turistas. Em sua última viagem, Bruce teve um desentendimento com o seu gerente de viagens e por isso quer comprar as passagens e reservar o hotel de maneira *online*. Bruce vai comprar as passagens e reservar o hotel através de um notebook antigo que tem guardado e que quase nunca utiliza.

Através do exemplo acima é possível deduzir um dos públicos alvos do site seriam adultos com idade próxima aos 50 anos e que são casados. Sabe-se que algum dos requisitos do site sejam, ter uma área para compra de passagens e outra para reserva em hotéis, e que todo o site deve ser rodado em um notebook simples e antigos.

## 2.8 Resumo do Capítulo

Para que se possa terceirizar o desenvolvimento de software dentro de Órgãos Pú-  
blicos, 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 es-  
tabelecidos pela Norma SQuaRE, quanto à categoria de manutenibilidade. Essa categoria  
é fundamental na contratação de software, pois permite saber o quanto manutenível é o soft-  
ware 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.

Aprendizagem de máquina tem por objetivo auxiliar o computador na tomada de decisões. 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. Ao fim do capítulo é apresentado o SUS que é um questionário de avaliação para avaliar o grau de usabilidade de um sistema.



# 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 GIT<sup>1</sup> 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 2013].

O motivo para escolha da ferramenta se deve ao fato de que o Git contém o suporte para desenvolvimento linear, o que garante um paralelismo de diversas áreas do desenvolvimento. Outro diferencial do Git está nos *snapshots* dos objetos que são armazenados. Isso significa que o Git não rearmazena arquivos que não foram alterados [MARTINHO e MUNIZ 2013].

### 3.1.2 Github

O Github<sup>2</sup> 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.

---

<sup>1</sup> <https://git-scm.com/>

<sup>2</sup> <https://github.com>

### 3.1.3 SonarQube

O SonarQube é uma ferramenta *open-source* de análise estática de código-fonte que foca em analisar sete ramos da qualidade de código, como pode ser visto na Figura 15. A ferramenta apresenta métricas quanto a duplicação de código, testes unitários, complexidade, *bugs* em potencial, regras da linguagem, comentários e arquitetura e *design* [SonarQube documentation]. A ferramenta foi escrita em Java e seu foco está em lidar com defeitos de código. Contudo, existe uma diversidade de *plugins* que estendem as funcionalidades da ferramenta [Ferenc 2014]. A arquitetura do SonarQube é composta



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

de quatro componentes, como pode ser visto na Figura 16 [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.

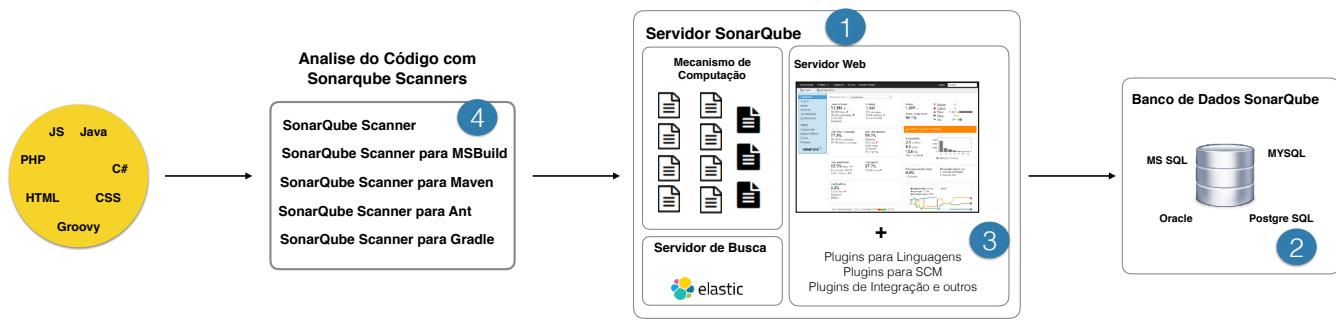


Figura 16 – Arquitetura SonarQube composta de quatro componentes: [SonarQube documentation]

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

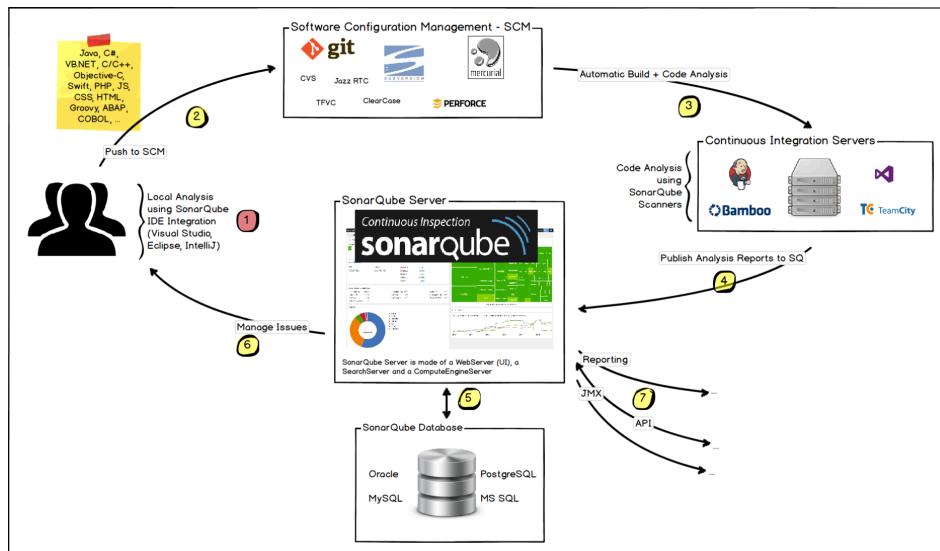


Figura 17 – Integração SonarQube em Diversas Áreas de Desenvolvimento de Software: [SonarQube documentation]

1. Os desenvolvedores podem utilizar uma instância do SonarQube (SonarLint) em suas próprias IDEs para rodar uma análise local.
2. Desenvolvedores podem subir seus códigos para suas ferramentas de *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 ( [Júnior e José Roberto 2010], [Fernandes 2005], [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 Bonita<sup>3</sup> 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 LaTeX<sup>4</sup> 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 3<sup>5</sup> 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

Zotero<sup>6</sup> 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

---

<sup>3</sup> <http://www.bonitasoft.com/>

<sup>4</sup> <https://www.latex-project.org/>

<sup>5</sup> <https://www.sublimetext.com/3>

<sup>6</sup> <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 18 apresenta o processo de desenvolvimento adotado pelo Órgão X.

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 ao processo de desenvolvimento do software. A Figura 19 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*, o Gestor de TI e a equipe de qualidade do Órgão, podem acompanhar o projeto. A figura 20 apresenta outra visão da integração entre a proposta e o ambiente de configuração do Órgão X.

## 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

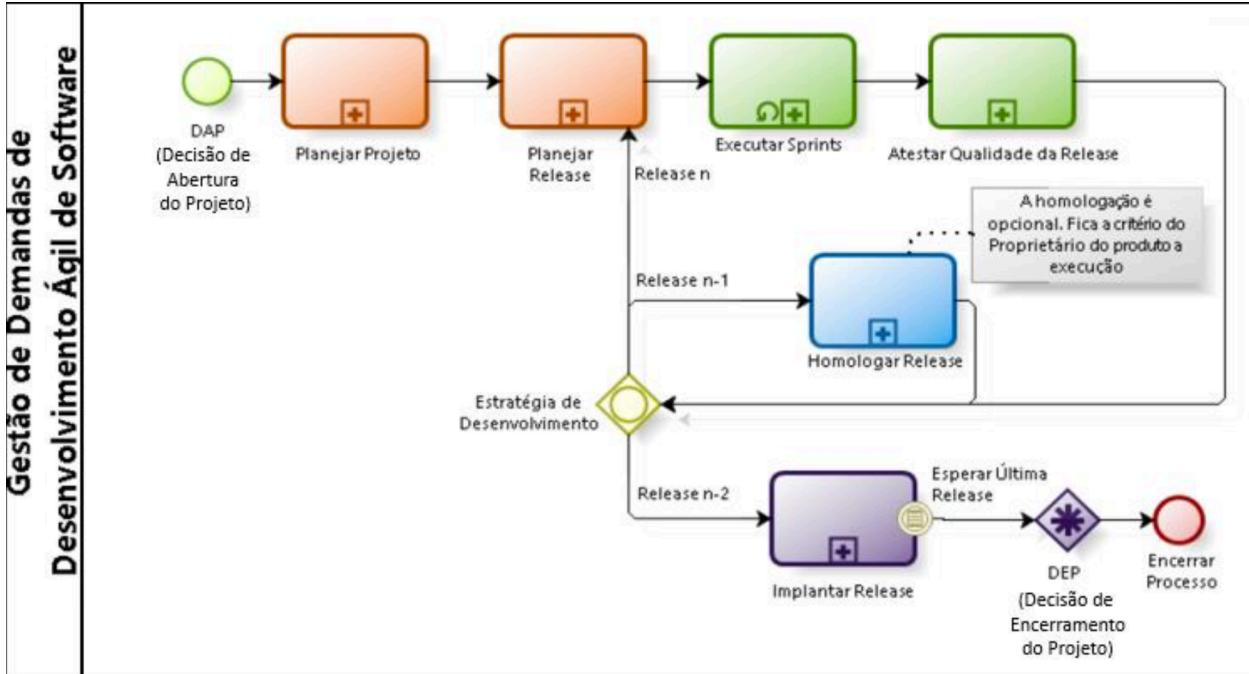


Figura 18 – Processo de Desenvolvimento Adotado Pelo Órgão X. Fonte: [Schaidt e Regis 2014]

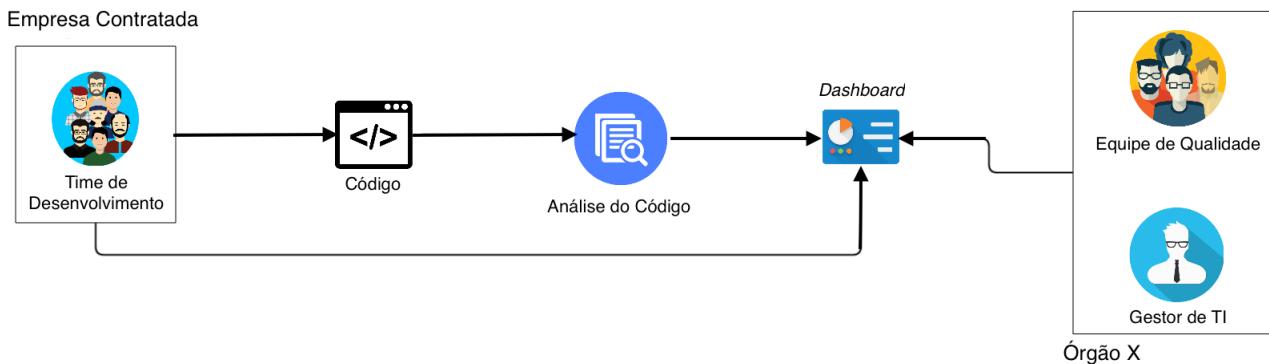


Figura 19 – Ciclo de Verificação Utilizando a Solução Compartilhada Entre Órgão Contratante e Empresa Contratada

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.

Essas métricas serão atribuídas ao longo do desenvolvimento do projeto, sendo assim, durante as primeiras *sprints* do projeto serão utilizadas métricas diferentes e mais relevantes ao andamento do projeto, das que as métricas que serão coletadas ao fim do

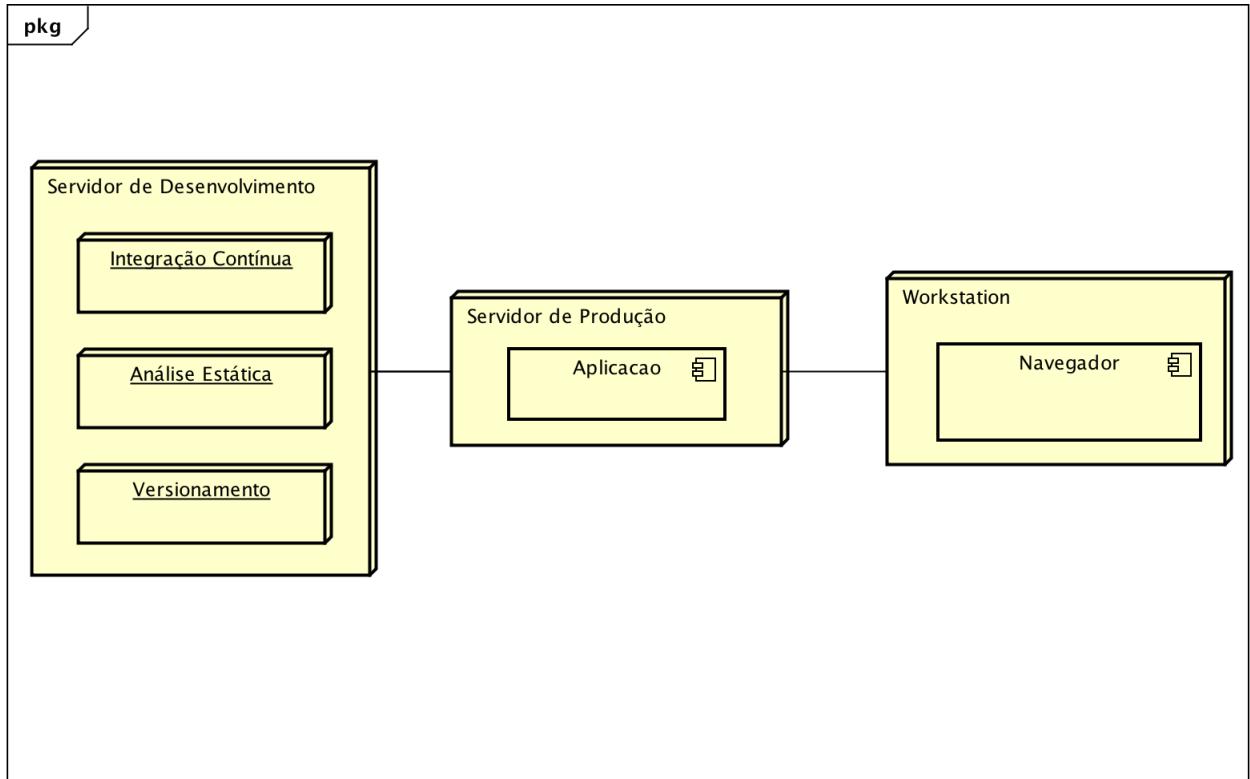


Figura 20 – Diagrama de Implantação da Solução no Órgão X

projeto. A escolha das métricas a serem utilizadas em cada momento do desenvolvimento será feita através de um questionário com até quatro gerentes de projeto e uma persona criada para simular um gestor "ideal". As respostas coletadas no questionário serão utilizadas como base na criação de perfis. Esse perfil servirá como um indicador para as métricas que serão sugeridas à futuros gestores. A figura 21 apresenta um exemplo com três possíveis perfis que podem ser coletados através das respostas dos gestores. Na figura percebe-se que o primeiro perfil prefere um conjunto de métricas na primeira *sprint* e ao longo do desenvolvimento do projeto ir acrescentando mais métricas para serem acompanhadas. Já o segundo perfil coleta determinadas métricas no início e nas *sprints* seguintes ele acompanha somente uma métrica específica. O terceiro perfil avalia um conjunto de métricas nas *sprints* ímpares e outro conjunto nas *sprints* pares.

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

	 <b>Perfil 1</b>	 <b>Perfil 2</b>	 <b>Perfil 3</b>
<b>Sprint 1</b>	<b>LOC</b> <b>CC</b> <b>% CT</b>	<b>LOC</b> <b>% CT</b>	<b>No de Defeitos</b> <b>% CT</b>
<b>Sprint 2</b>	<b>LOC</b> <b>CC</b> <b>% CT</b> <b>DIT</b>	<b>MHF</b>	<b>No de Defeitos</b> <b>% CT</b> <b>CC</b> <b>LCOM</b>
<b>Sprint 3</b>	<b>LOC</b> <b>CC</b> <b>% CT</b> <b>DIT</b> <b>LCOM</b>	<b>MHF</b>	<b>No de Defeitos</b> <b>CC</b> <b>CFA</b>
<b>Sprint 4</b>	<b>LOC</b> <b>CC</b> <b>LCOM</b> <b>% CT</b> <b>DIT</b>	<b>MHF</b>	<b>DIT</b> <b>% CT</b> <b>LCOM</b>

Figura 21 – Exemplo de Perfis Coletados Após Questionário

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.

Foi elaborado uma prova de conceito, para que se pudesse testar a funcionalidade relacionada à coleta de métricas. Esta prova, consiste em, elaborar um software que fosse capaz de extrair métricas do relatório do SonarQube. Inicialmente foi necessário criar uma instância do SonarQube e fazer a análise de um projeto. O projeto escolhido foi um projeto em Java, que o próprio SonarQube disponibiliza para download, com o intuito de ser utilizado como exemplo. Com o ambiente configurado, implementou-se uma solução em Python que atendesse aos requisitos estabelecidos para a prova de conceito. A Figura 22 apresenta a saída do console, ao se rodar a solução.

```

~/Documents/TCC1/Software — -bash
[MBP-de-Levi:Software levimoraes$ python codigo_sonar.py
[<span id="m_ncloc">31</span>
[<span id="m_complexity">16</span>
[<span id="m_comment_lines_density">0.0%</span>
[<span id="m_duplicated_lines_density">48.6%</span>
[<span id="m_blocker_violations">0</span>
[<span id="m_critical_violations">0</span>
[<span id="m_major_violations">4</span>
[<span id="m_minor_violations">6</span>
MBP-de-Levi:Software levimoraes$
```

Figura 22 – Métricas Extraídas do SonarQube

Na primeira linha da Figura 22, é executada a chamada do código. As linhas seguintes apresentam as métricas coletadas pela solução. As métricas que são apresentadas no console, foram definidas no código, por este motivo, ainda não é possível definir outras métricas durante a execução da aplicação, esta funcionalidade será implementada na segunda fase do projeto.

### 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 23 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.

A página inicial da solução, também chamada de *home page*, aqui apresentada na Figura 24. 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 visua-

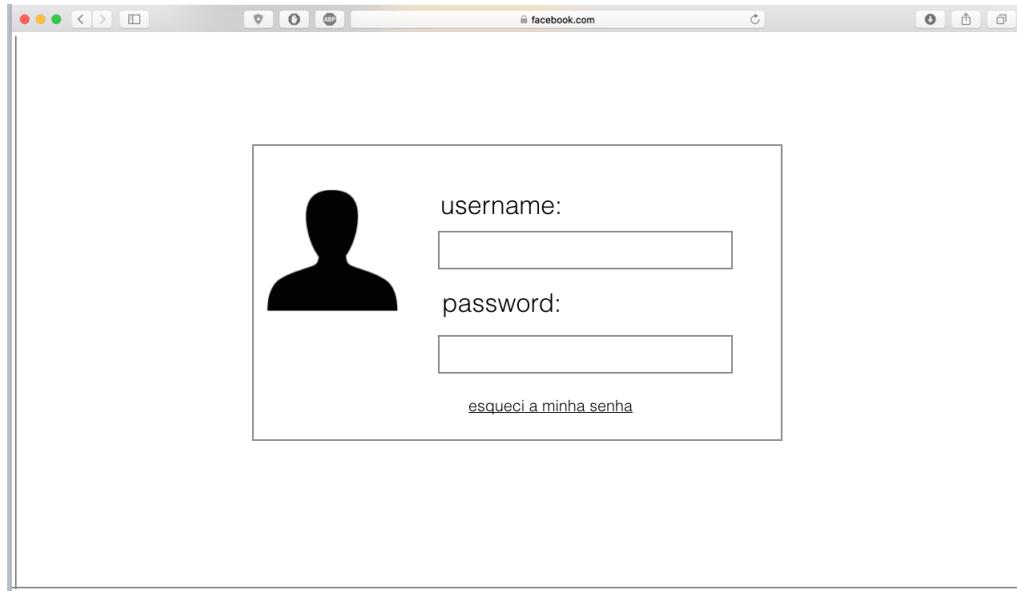


Figura 23 – Tela Login



Figura 24 – Tela Home

lizadas, 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.

A Figura 25 apresenta um protótipo do *dashboard* que será implementado. Nesta

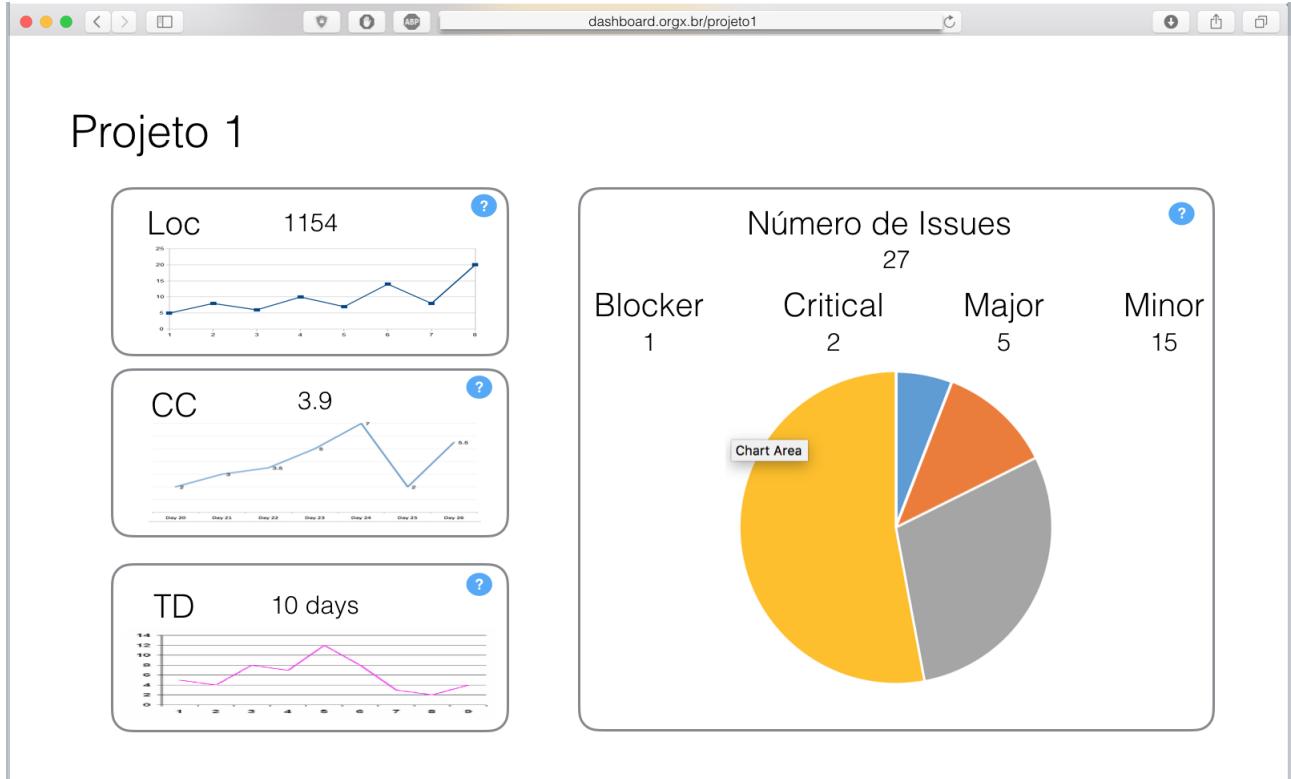


Figura 25 – Tela Dashboard

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 26 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.

Assim como a coleta de métricas, também foi elaborada uma prova de conceito, para testar a exibição dos gráficos que seriam mostrados no *dashboard*. Esta era uma continuação da prova de conceito da coleta de métricas, em que, feita a coleta, deveria-se exibir as métricas em um gráfico. A Figura 27 apresenta a tela relacionada à esta prova de conceito.

Nome do Projeto:

Descrição:

url:

Métricas:

- LOC ?
- TD ?
- CC ?
- Cobertura ?
- Duplicação ?
- Usar Edital ?

Salvar

Cancelar

Figura 26 – Tela Adicionar

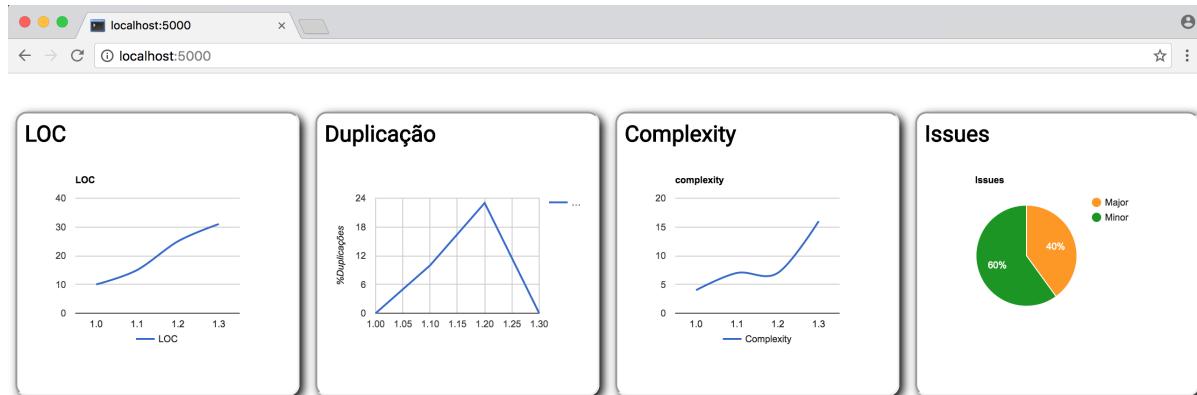


Figura 27 – Página do Dashboard da Prova de Conceito

## 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 tal experiência com contratação de software para Órgãos Públcos, 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 ??

Uma segunda parte da avaliação se refere a usabilidade do software desenvolvido. Para fazer esta avaliação será utilizado o modelo de questionário SUS

## 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.



# 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 28 apresenta em quais classificações esse trabalho se baseia.



Figura 28 – 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 29) 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.

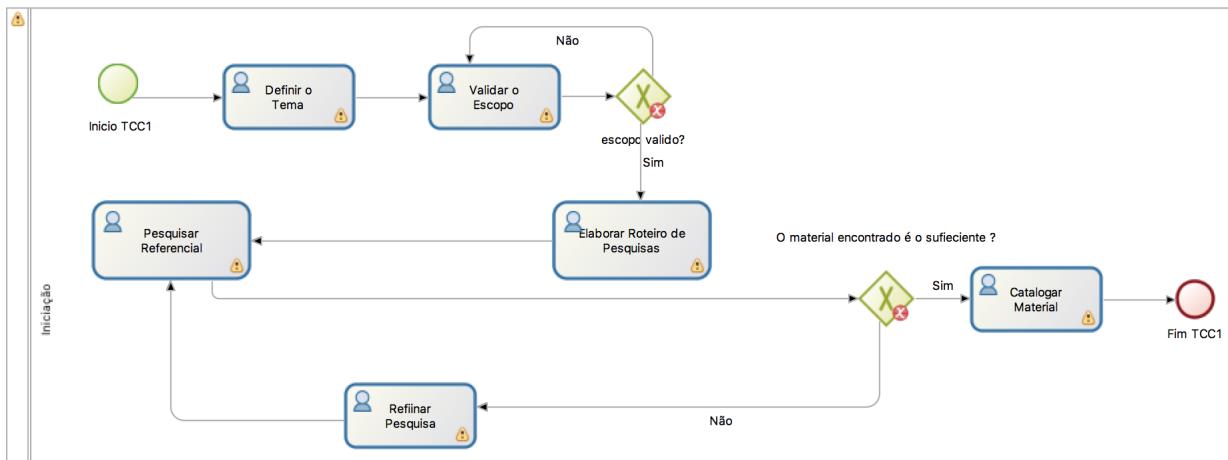


Figura 29 – Modelagem da Fase de Iniciação

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 30.

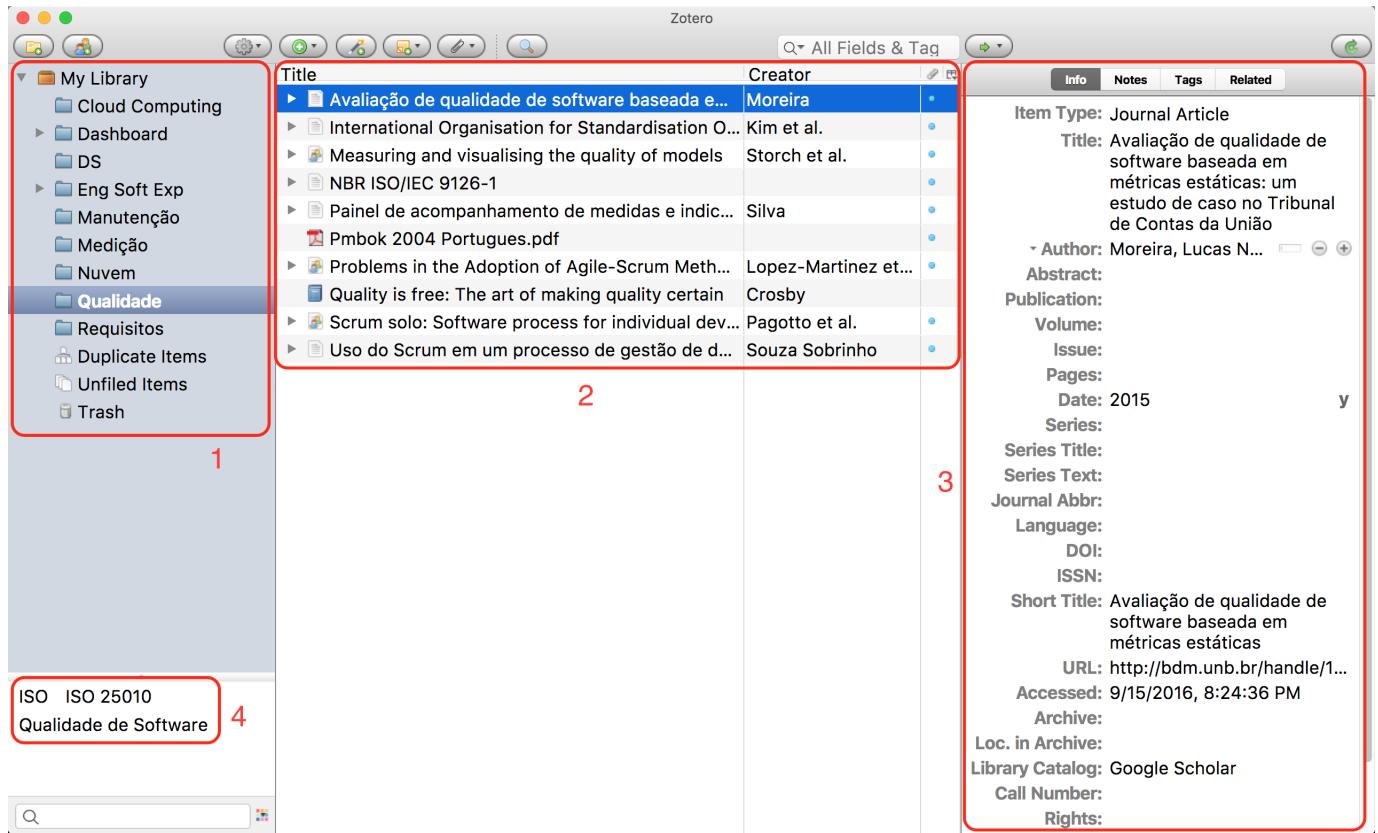


Figura 30 – *Screenshot* do Zotero contendo as categorias dos materiais pesquisados, suas tags e anotações

- 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 31) 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

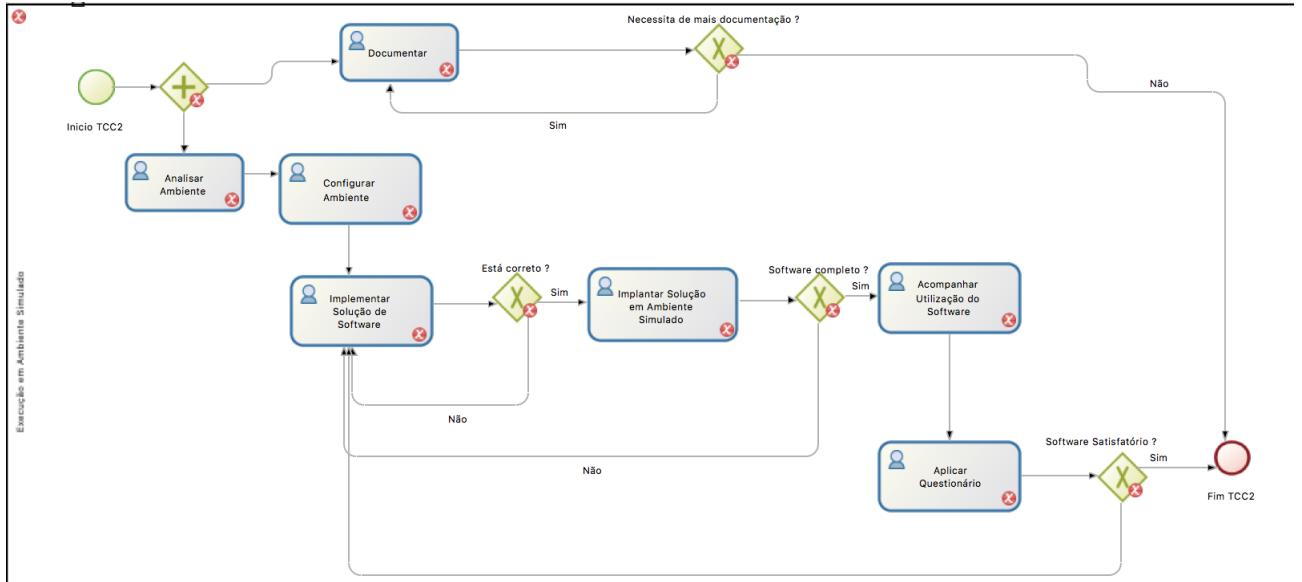


Figura 31 – Modelagem da Fase de Execução

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 32 apresenta uma visão geral do processo que será aplicado neste trabalho.

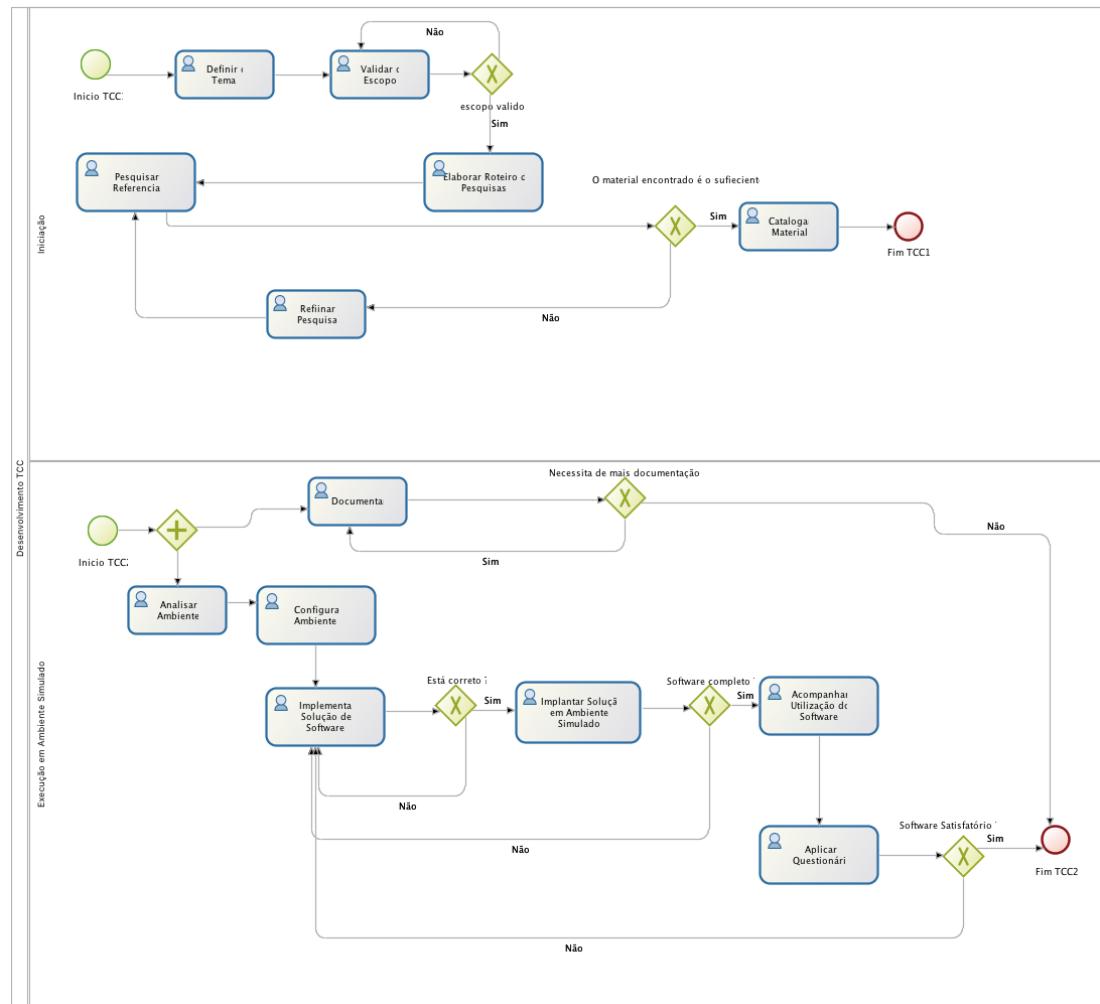


Figura 32 – Plano de Pesquisa

## 5.2 Metodologia de Desenvolvimento

O *dashboard* será criado utilizando uma adaptação da metodologia de desenvolvimento Scrum [Pagotto 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 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 2016]. Na Figura 33 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].

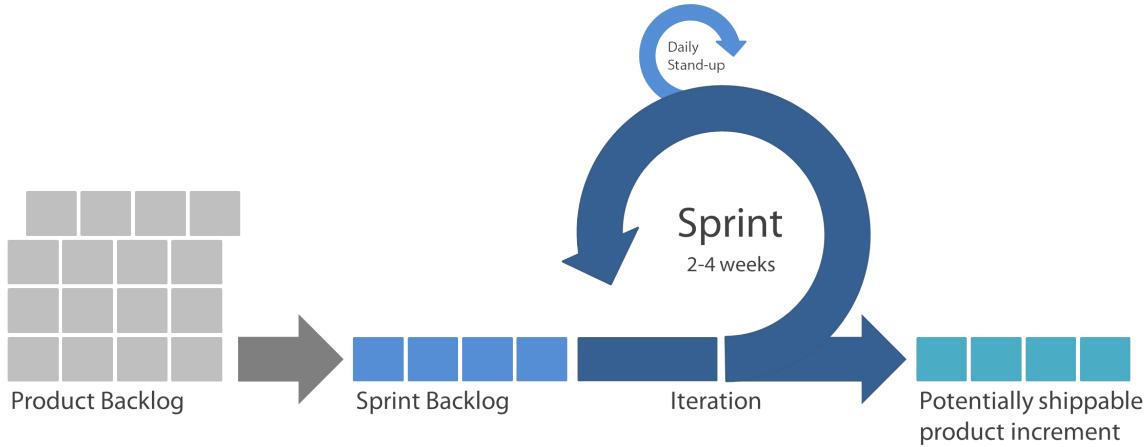


Figura 33 – Ciclo de Desenvolvimento do Scrum

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 34. A confirmação ou critérios de aceitação é um *checklist* abordando o que deve ser verificado para que a história seja dada como concluída.

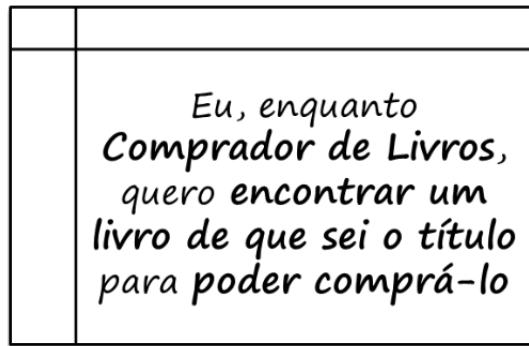


Figura 34 – Exemplo de História de Usuário. Fonte: [Sabbagh 2014]

- **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 35.

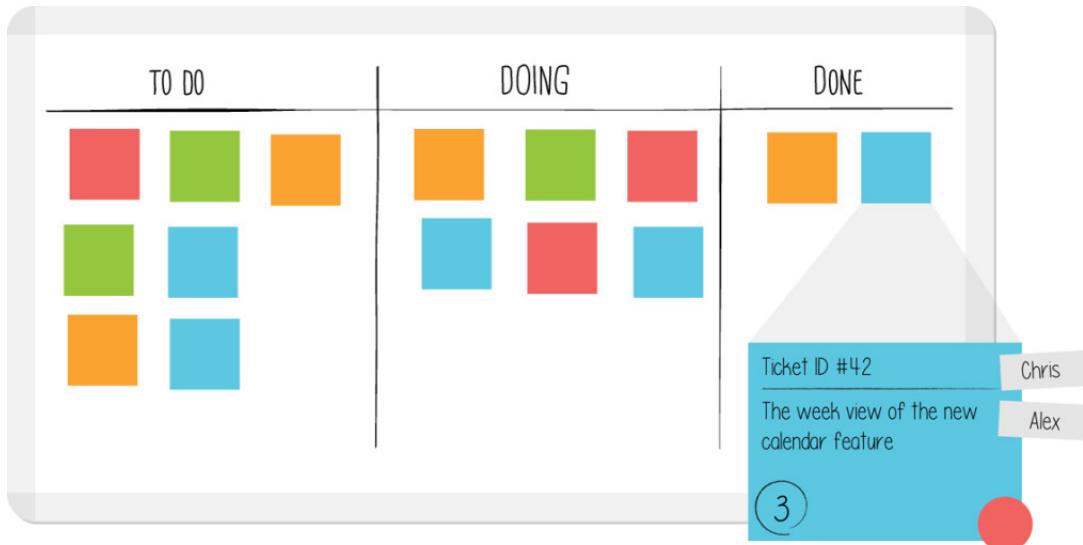


Figura 35 – Exemplo de *Kanban*

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 5 e Tabela 6) referentes às atividades do TCC1 e TCC2 respectivamente.

### 5.3.1 Cronograma TCC1

A Tabela 5 apresenta um cronograma em relação às atividades a serem desenvolvidas enquanto TCC1.

Tabela 5 – 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 6 apresenta um cronograma em relação às atividades a serem desenvolvidas enquanto TCC2.

Tabela 6 – 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 7 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 7 – *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. 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 47.
- BRASIL. *Lei n 8.666, de 21 de junho de 1993*. 1993. Citado na página 23.
- BRASIL. *DECRETO No 2.271, DE 7 DE JULHO DE 1997*. [S.l.], 1997. Citado 2 vezes nas páginas 19 y 23.
- BRASILL (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 23.
- BROOKE, J. et al. Sus-a quick and dirty usability scale. *Usability evaluation in industry*, London, United kingdom, v. 189, n. 194, p. 4–7, 1996. Citado na página 43.
- BRUNIALTI, L. F. et al. aprendizado de máquina em sistemas de recomendação baseados em conteúdo textual uma revisão sistemática. In: *XI Brazilian Symposium on Information System. Goiania, GO: 2015* Disponível em:< <http://www.lbd.dcc.ufmg.br/colecoes/sbsi/2015/029.pdf>>. Acesso em. [S.l.: s.n.], 2015. v. 5. Citado na página 35.
- 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 29.
- CHARTSJS documentation. <<http://www.chartjs.org/docs/>>. Accessed: 2016-10-10. Citado na página 50.
- 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 31.
- CROSBY, P. B. *Quality is free: The art of making quality certain*. [S.l.]: Signet, 1980. Citado na página 20.
- 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](http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=988358)>. Citado na página 37.
- FERENC, R. Source Meter Sonar Qube Plug-in. In: . [S.l.]: IEEE, 2014. p. 77–82. Citado na página 48.
- FERNANDES, J. U. J. Licitação na modalidade pregão, na forma eletrônica-Regulamentação: Decreto n. 5.450, de 31 de maio de 2005. *Fórum de Contratação e Gestão Pública [recurso eletrônico]*, 2005. Disponível em: <<https://dspace.almg.gov.br/handle/11037/6020>>. Citado na página 50.
- FEW, S. *Information Dashboard Design - The effective Visual Comunication of Data*. [S.l.]: O'Reilly, 2006. Citado 7 vezes nas páginas 11, 36, 37, 39, 40, 41 y 42.

- FILHO, C. F. *História da computação: O Caminho do Pensamento e da Tecnologia.* [S.I.]: EDIPUCRS, 2007. Citado na página 19.
- GERHARDT, T. E.; Silveira , D. T. *Métodos de pesquisa.* [S.I.]: 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 na página 36.
- GOOGLE Charts documentation. <<https://developers.google.com/chart/interactive/docs/reference>>. Accessed: 2016-10-10. Citado na página 50.
- 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 na página 36.
- HASAN, H.; ABDUL-KAREEM, S. Human–computer interaction using vision-based hand gesture recognition systems: a survey. *Neural Computing and Applications*, v. 25, n. 2, p. 251–261, ago. 2014. ISSN 0941-0643, 1433-3058. Disponível em: <<http://link.springer.com/10.1007/s00521-013-1481-0>>. Citado na página 36.
- ISO 25010. *Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models.* 2011. ed. [S.I.]. Citado 2 vezes nas páginas 11 y 29.
- JANNACH, D. et al. *Recommender systems: an introduction.* [S.I.]: Cambridge University Press, 2010. Citado na página 35.
- JÚNIOR, T.; José Roberto. Adesão à ata de registro de preços: o carona que virou inexigibilidade. *Fórum de contratação e gestão pública*, 2010. Disponível em: <<http://bdjur.stj.jus.br/dspace/handle/2011/32413>>. Citado na página 50.
- KANER, C. Software engineering metrics: What do they measure and how do we know? *10TH INTERNATIONAL SOFTWARE METRICS SYMPOSIUM*, 2004. Citado na página 31.
- KUSTERS, R. J.; HEEMSTRA, F. J. Software maintenance: an approach towards control. In: *Software Maintenance, 2001. Proceedings. IEEE International Conference on.* [S.I.: s.n.], 2001. p. 667–670. ISSN 1063-6773. Citado na página 30.
- 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](http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=7177443)>. Citado na página 39.
- LOPEZ-MARTINEZ, J. 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 19.
- MAHNIC, V. A case study on agile estimating and planning using scrum. *Elektronika ir Elektrotechnika*, v. 111, n. 5, p. 123–128, 2011. Disponível em: <<http://ecoman.ktu.lt/index.php/elt/article/view/372>>. Citado na página 69.
- MANESS, J. M.; MIASKIEWICZ, T.; SUMNER, T. Using personas to understand the needs and goals of institutional repository users. *D-Lib Magazine*, v. 14, n. 9, p. 10, 2008. Citado na página 43.
- MANNING, H.; TEMKIN, B.; BELANGER, N. The power of design personas. *Cambridge, MA: Forrester Research*, 2003. Citado na página 44.
- 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](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 47.
- 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 31 y 33.
- MINERAL, D. N. de P. *Edital Pregão Eletronico No 14/2015*. [S.l.], 2015. Citado 5 vezes nas páginas 13, 25, 50, 54 y 57.
- MITCHELL, T. M.; LEARNING, M. McGraw-hill science. *Engineering/Math*, v. 1, 1997. Citado na página 34.
- MONARD, M. C.; BARANAUSKAS, J. A. Conceitos sobre aprendizado de máquina. *Sistemas Inteligentes-Fundamentos e Aplicações*, v. 1, n. 1, 2003. Citado na página 34.
- MOREIRA, L. N. Avaliação de qualidade de software baseada em métricas estáticas: um estudo de caso no Tribunal de Contas da União. 2015. Disponível em: <<http://bdm.unb.br/handle/10483/10107>>. Citado na página 33.
- 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](http://ftp.unisc.br/portal/upload/com_arquivo/1370886616.pdf)>. Citado 3 vezes nas páginas 12, 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 11, 26 y 27.
- NUNNALLY, J. C. (1978). *Psychometric theory*, v. 2, 1978. 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 19 y 26.

PAGOTTO, T. 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.

PERES, S. M. et al. Tutorial sobre fuzzy-c-means e fuzzy learning vector quantization: Abordagens híbridas para tarefas de agrupamento e classificação. *Revista de Informática Teórica e Aplicada*, v. 19, n. 1, p. 120–163, 2012. Citado na página 34.

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 11, 30 y 31.

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

PRESSMAN, R. S. *Software Engineering - A Practitioner's Approach*. 7. ed. [S.l.]: Higher Education, 2010. Citado 2 vezes nas páginas 29 y 31.

PRUITT, J.; ADLIN, T. *The persona lifecycle: keeping people in mind throughout product design*. [S.l.]: Morgan Kaufmann, 2010. Citado na página 43.

RECOMMENDER systems-How they works and their impacts. 2006. Disponível em: <<http://findoutyourfavorite.blogspot.com.br/2012/04/content-based-filtering.html>>. Citado 2 vezes nas páginas 11 y 36.

RUSSELL, S. Artificial intelligence: A modern approach author: Stuart russell, peter norvig, publisher: Prentice hall pa. 2009. Citado na página 34.

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%B3es.+Hoje+%C3%A9+poss%C3%ADvel+ver+os+assuntos+ligados%22+%22Mas,+segundo+o+pr%C3%ADrio+Rafael%22+%22que+n%C3%A3o+%C3%A9+necess%C3%A1rio+ter+uma+ampla+compreens%C3%A3o+do+Scrum+para%22+&ots=ERSywNIEAd&sig=rvmJD\\_BXcQY-P0MYZBqQJOaOcOA](https://books.google.com/books?hl=en&lr=&id=pG-CCwAAQBAJ&oi=fnd&pg=PT9&dq=%22Agilidade+em+diferentes+organiza%C3%A7%C3%B3es.+Hoje+%C3%A9+poss%C3%ADvel+ver+os+assuntos+ligados%22+%22Mas,+segundo+o+pr%C3%ADrio+Rafael%22+%22que+n%C3%A3o+%C3%A9+necess%C3%A1rio+ter+uma+ampla+compreens%C3%A3o+do+Scrum+para%22+&ots=ERSywNIEAd&sig=rvmJD_BXcQY-P0MYZBqQJOaOcOA)>. Citado 4 vezes nas páginas 12, 68, 69 y 70.

SARWAR, B. et al. Item-based collaborative filtering recommendation algorithms. In: ACM. *Proceedings of the 10th international conference on World Wide Web*. [S.l.], 2001. p. 285–295. Citado 2 vezes nas páginas 11 y 35.

SAURO, J.; LEWIS, J. R. *Quantifying the user experience: Practical statistics for user research*. [S.l.]: Morgan Kaufmann, 2016. Citado 3 vezes nas páginas 13, 42 y 43.

SCHAIDT, L.; Regis, Y. *Monitoramento de Qualidade de Software na Administração Pública Federal*. Dissertação (Mestrado) — Universidade de Brasília, 2014. Citado 4 vezes nas páginas 11, 28, 53 y 54.

SCHNAIDER, L. Uma abordagem para Medição e Análise em Projetos de Desenvolvimento de Software. *Brasília, SBQS*, 2004. Citado na página 20.

SCHWENDIMANN, B. 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 [11](#) y [37](#).

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 [53](#).

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

SONARQUBE documentation. <<http://docs.sonarqube.org/display/SONAR/Architecture+and+Integration>>. Accessed: 2016-10-10. Citado 3 vezes nas páginas [11](#), [48](#) y [49](#).

TCU. *Guia de boas práticas em contratação de soluções de tecnologia da Informação*. [S.l.], 2012. Citado na página [23](#).



# Apêndices



# APÊNDICE A – Formulário de Consentimento

The screenshot shows a web-based consent form template from usability.gov. At the top, the usability.gov logo is displayed with the tagline "Improving the User Experience". The main title of the form is "Consent Form (Adult)".  
**I agree to participate in the study conducted by the Universidade de Brasília.**  
I understand that participation in this usability study is voluntary and I agree to immediately raise any concerns or areas of discomfort during the session with the study administrator.  
Please sign below to indicate that you have read and you understand the information on this form and that any questions you might have about the session have been answered.  
**Date:** \_\_\_\_\_  
**Please print your name:** \_\_\_\_\_  
**Please sign your name:** \_\_\_\_\_  
**Thank you!**  
We appreciate your participation.

At the bottom, there is a footer with the text "U.S. Department of Health & Human Services - 200 Independence Avenue, S.W. - Washington, D.C. 20201" and social media icons for email and Twitter.

Figura 36 – Formulário de Consentimento