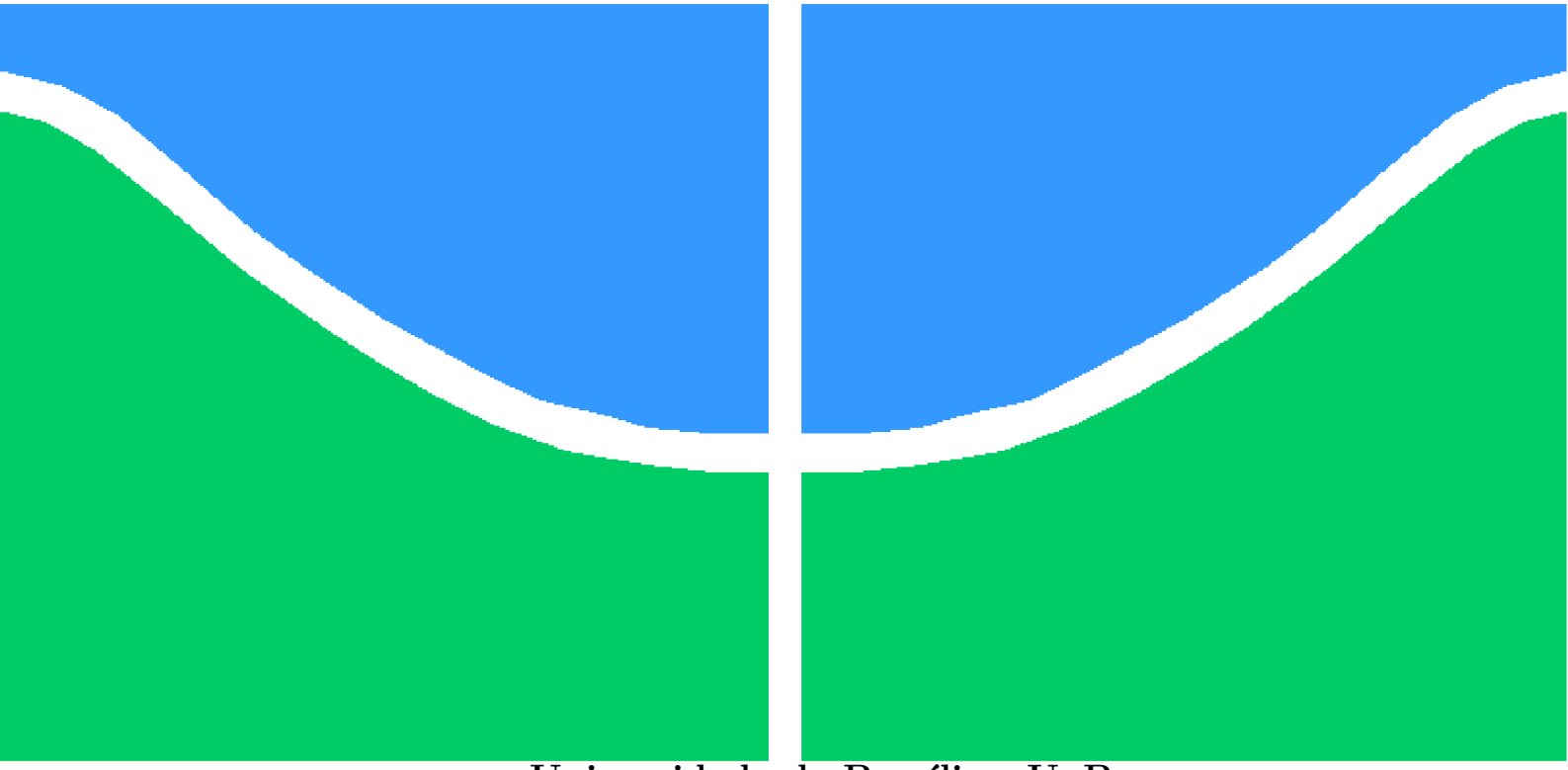

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Engenharia de Software

Estudo da Eficácia e Eficiência do Uso de um Ambiente de Data Warehousing para Aferição da Qualidade Interna de Software: um Estudo de Caso Preliminar na Caixa Econômica Federal

Autor: Nilton Cesar Campos Araruna

Orientador: Prof. Msc. Hilmer Rodrigues Neri

Brasília, DF

2014

Nilton Cesar Campos Araruna

Estudo da Eficácia e Eficiência do Uso de um Ambiente de Data Warehousing para Aferição da Qualidade Interna de Software: um Estudo de Caso Preliminar na Caixa Econômica Federal

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. Msc. Hilmer Rodrigues Neri

Brasília, DF

2014

Nilton Cesar Campos Araruna

Estudo da Eficácia e Eficiência do Uso de um Ambiente de Data Warehousing para Aferição da Qualidade Interna de Software: um Estudo de Caso Preliminar na Caixa Econômica Federal/ Nilton Cesar Campos Araruna. – Brasília, DF, 2014-

86 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Msc. Hilmer Rodrigues Neri

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2014.

1. Métricas de Código-Fonte. 2. *Data Warehousing*. I. Prof. Msc. Hilmer Rodrigues Neri. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Estudo da Eficácia e Eficiência do Uso de um Ambiente de Data Warehousing para Aferição da Qualidade Interna de Software: um Estudo de Caso Preliminar na Caixa Econômica Federal

CDU a obter

Nilton Cesar Campos Araruna

Estudo da Eficácia e Eficiência do Uso de um Ambiente de Data Warehousing para Aferição da Qualidade Interna de Software: um Estudo de Caso Preliminar na Caixa Econômica Federal

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, 26 de Novembro de 2014:

Prof. Msc. Hilmer Rodrigues Neri
Orientador

Profa. Milene Serrano
Convidado 1

Prof. Luiz Augusto Fontes Laranjeira
Convidado 2

Brasília, DF
2014

Este trabalho é dedicado ao meu pai, Aurenilton Araruna, minha mãe, Cláudia Araruna e ao meu irmão, Gustavo Campos. Estas inseriram na minha vida as virtudes do esforço.

Agradecimentos

Agradeço primeiramente aos meus pais, Aurenilton e Claudia, por sempre acreditarem em mim e no meu potencial, por me darem apoio, força e amor sempre que necessário, por me proporcionarem uma vida excelente cheia de conforto e por não medirem esforços na hora de me amparar. Obrigado por me ensinarem os grandes valores da vida, do valor do esforço, do tempo e do dinheiro. Vocês sempre serão meus heróis e minhas grandes referências.

Agradeço ao meu irmão, Gustavo Henrique, que sempre me mostrou o que é ser um homem intelectual e de caráter, mesmo com tamanhas diferenças entre nós, sempre me serviu como modelo. Se não fossem suas conquistas não acreditaria que as minhas seriam possíveis.

Agradeço a minha namorada, Giovana Giugliani, que esteve ao meu lado em toda essa fase da minha vida, do cursinho até a graduação, vivenciando todo meu esforço, minhas dúvidas e minha ansiedade. Agradeço pelo apoio, pela paciência, pela ajuda, pela companhia em noites de estudo e as tornando bem melhor, também peço desculpa pelo estresse e pelo tempo que tive que me ausentar do seu lado.

Agradeço ao meu orientador Hilmer Rodrigues Neri pela confiança depositada em mim, pelo aprendizado dentro e fora de sala e por ter me motivado ao longo da graduação. Agradeço aos meus amigos e colegas Matheus e Pedro Tomioka pela colaboração e ajuda não só durante a execução desse trabalho mas também em toda a graduação.

Agradeço a toda a minha família, avós, avôs, madrinha, padrinho, tios, tias, primos, primas e aos meus amigos de infância, vulgo frangos, por entenderem a minha ausência durante essa fase da minha vida e por sempre desejarem a minha graduação e um belo futuro profissional para poder pagar a conta do bar.

Agradeço a equipe PEDeS da CAIXA, onde tive o prazer de estagiar, em especial ao meu supervisor, Diego Costa, pela oportunidade, pelo aprendizado que levarei pelo resto da vida, por me prepararem para o mercado de trabalho, por me ensinarem tudo o que sei sobre ser um profissional e pelos momentos de alegria. Sei que aprendi com os melhores.

Agradeço ao grupo kanburn, Thiago Kairala, Bruno Contessoto, Rafael Fazzolino, Eduardo Brasil, Thabata Granja e em especial ao meu primo João Araruna que ingressou na universidade junto comigo e participou da minha luta, pelos encontros regados de estudos, pizza e alegria. Os melhores e mais chatos amigos que não escolhi viver mas fui obrigado e hoje a obrigação é manter cada um na minha vida.

"Learn from yesterday, live for today, hope for tomorrow. The important thing is not to stop questioning."

(Albert Einstein)

Resumo

A qualidade do software depende da qualidade do código-fonte, um bom código-fonte é um bom indicador de qualidade interna do produto de *software*. Portanto, o monitoramento de métricas de código-fonte de um *software* significa melhorar a sua qualidade. Existem diversas soluções e ferramentas para se obter um monitoramento de métricas de *software* e que conseguem extrair valores de métricas de código com facilidade. Porém, a decisão sobre o que fazer com os dados extraídos ainda é uma dificuldade relacionada à visibilidade e interpretação dos dados. Este trabalho se propõe a analisar a eficácia e eficiência do uso de um ambiente de *Data Warehousing(DW)* para facilitar a interpretação, visibilidade e avaliação das métricas de código-fonte, associando-as a cenários de limpeza. Os cenários de limpeza buscam apoiar as tomadas de decisão que reflitam na alteração do código-fonte e um *software* com mais qualidade. Para um melhor entendimento da solução DW proposta e dos elementos que dizem respeito a sua arquitetura e seus requisitos de negócio foram apresentas as fundamentações teóricas necessárias. Um projeto para a realização de uma investigação empírica foi elaborado utilizando a técnica de estudo de caso. O projeto visa responder questões qualitativas e quantitativas a respeito da eficácia e eficiência na utilização da solução citada em um órgão público.

Palavras-chaves: Métricas de Código-Fonte. *Data Warehousing*. *Data Warehouse*. Eficácia. Eficiência

Abstract

The quality of the software depends on the quality of the source code, a good source code is a good indicator of the software internal quality. Therefore, the monitoring of metrics of a software source-code results in improving its quality. There are several solutions and tools to achieve software monitoring of metrics that can easily extract values of metrics of code. However, the decision regarding what to do with the extracted data is still an issue related to the visibility and interpretation of this data. The purpose of this project is to analyse the effectiveness and efficiency of the use of the environment Data Warehousing to facilitate interpretation, visibility and evaluation of source code metrics, associating them with cleansing scenarios. The cleansing scenarios are to support decision making processes that reflect on the alteration of the source code and a software with greater quality. For a better understanding of the DW solution proposedand of the elements that concern its arquitecture and its business requirements, the necessary theoretical fundaments have been presented. The project aims to answer qualitative and quantitative questions about the concerning the public.

Key-words: Source Code Metrics, Data Warehousing, Data Warehouse, Effectiveness, efficiency

Lista de ilustrações

*

Lista de tabelas

*

Lista de abreviaturas e siglas

ACC	*Afferent Connections per Class*
ACCM	*Average Cyclomatic Complexity per Method*
AMLOC	*Average Method Lines of Code*
ANPM	*Average Number of Parameters per Method*
CBO	*Coupling Between Objects*
CETEC	*Centralizadora Nacional de Tecnologia da Informação*
CSV	*Comma-Separated Values*
DIT	*Depth of Inheritance Tree*
DW	*Data Warehouse*
ETL	*Extraction-Transformation-Load*
GITECBR	*Gerencia de Filial de Suporte Tecnológico de Brasília*
GQM	*Goal-Question-Metric*
IEC	*International Electrotechnical Commission*
ISO	*International Organization for Standardization*
LCOM4	*Lack of Cohesion in Methods*
LOC	*Lines of Code*
MCTI	*Modelo de Contratações de Soluções de TI*
NPA	*Number of Public Attributes*
NOC	*Number of Children*
NOM	*Number of Methods*
OLAP	*On-Line Analytical Processing*
OLTP	*Online Transaction Processing*
RFC	*Response For a Class*

SCAM	*IEEE International Working Conference on Source Code Analysis and Manipulation*
SISP	*Sistema de Administração de Recursos de Tecnologia da Informação e Informática*
S&SC	*Software e Serviços Correlatos*
XML	*Extensible Markup Language*

Sumário

*

1 Introdução

1.1 Contexto

Segundo(WILLCOCKS; LACITY, 2001) a terceirização do desenvolvimento de software é uma prática cada vez mais adotada por organizações. A terceirização de atividades, ou seja, o ato de transferir para fora da organização uma parte do seu processo produtivo não é uma prática recente. Atividades e processos que eram muito específicos dentro de uma organização foram transferidos, de forma parcial ou total, para outras organizações ou agentes externos (LEITE, 1997).

Uma das motivações para a terceirização é a qualidade do serviço prometida pelas empresas fornecedoras. No entanto, existem vários riscos associados à decisão pela terceirização, que podem comprometer a qualidade esperada, como por exemplo: as expectativas de serviço e a resposta rápida não serem atendidas adequadamente; o serviço prestado apresentar qualidade inferior ao existente anteriormente; e as tecnologias utilizadas não corresponderem ao esperado(WILLCOCKS; LACITY, 2001). Segundo SOFTEX (2013a) a aquisição de um *software* é um processo complexo, principalmente no que diz respeito à caracterização dos requisitos necessários ao *software* e às condições envolvidas na contratação como a qualidade esperada.

Acompanhando o ritmo da terceirização o cenário com empresas terceirizadas contratadas para o desenvolvimento de *software* está cada vez maior em organizações públicas. Tais organizações não são diretamente responsáveis pelo desenvolvimento do *software* mas são responsáveis pelo processo de verificação de sua qualidade conforme a norma (IN4, 2014) que será explicada no capítulo 5 deste trabalho.

1.2 Justificativa

Segundo (BECK, 1999)(FOWLER, 1999), a qualidade de *software* é medida pela qualidade de seu código-fonte. Conforme a (ISO/IEC 15939, 2002), medição é uma ferramenta primordial para avaliar a qualidade dos produtos e a capacidade de processos organizacionais, portanto, o Órgão Público contratante pode fazer uso do monitoramento de métricas de código-fonte para assistir ao processo de aferição de qualidade do *software* desenvolvido pela contratada.

Rêgo (2014) propôs uma solução para o monitoramento de métricas de código-fonte com suporte de um ambiente de *Data Warehousing* que será explicada no capítulo 5. Uma das principais contribuições deste trabalho foi a adição das análises de bugs e

violações à proposta de Rêgo (2014) e a elaboração de *dashboards*. O propósito destes incrementos será agregar valor para a aferição de qualidade de software.

1.3 Problema

Com foco na avaliação de controles gerais de Tecnologia da Informação nos Órgãos públicos, em 2011, o Tribunal de Contas da União-TCU detectou, por meio de auditorias de governança de TI em diversos Órgãos, uma considerável frequência de irregularidades relacionadas à inexistência, deficiências e a falhas de processos de *software* que comprometem a eficácia e eficiência das contratações de desenvolvimento de sistemas (BRASIL, 2011).

A inexistência de parâmetros de aferição de qualidade para contratação de desenvolvimento de sistemas e a deficiência no processo de contratação, decorrente da inexistência de metodologia que assegure boa contratação de desenvolvimento de sistemas foram listados nas auditorias como consequências da inexistência e falha de processos de *software* nos Órgãos públicos. Os critérios indicados pelo TCU no acórdão (BRASIL, 2011) foram : Constituição Federal, art. 37, caput; Instrução Normativa 4/2008, SLTI/MPOG, art. 12, inciso II; Lei 8666/1993, art. 6º, inciso IX; Norma Técnica - ITGI - Cobit 4.1, PO8.3 - Padrões de desenvolvimento e de aquisições; Norma Técnica - NBR ISO/IEC - 12.207 e 15.504;e Resolução 90/2009, CNJ, art. 10.

A auditoria do acordão (BRASIL, 2011) reforça o problema da falta de capacidade da administração pública de aferir a qualidade interna dos produtos de software desenvolvido por terceirizadas. A partir desse problema e da solução para monitoramento de métricas de código-fonte com suporte de um ambiente de *Data Warehousing-DWing* proposta por Rêgo (2014), foi formulada a questão de pesquisa geral deste trabalho que é:

O uso de um ambiente de DW para aferição da qualidade interna do software apoiado pela análise de cenários de limpeza, análise de bugs e análise de violações pode assistir ao processo de aferição da qualidade de software?

1.4 Objetivos

O objetivo geral deste trabalho é realizar um estudo de caso onde será analisado o uso de um ambiente de *Data Warehousing* para o monitoramento de métricas, bugs e violações de código fonte de forma a assistir ao processo de aferição de qualidade interna dos produtos de software desenvolvidos por uma organização terceirizada e adquiridas por uma organização pública. Dentre os objetivos específicos deste trabalho estão:

- Adicionar a análise de bugs utilizando a ferramenta findbugs à solução de um ambiente de *Data Warehousing-DWing* proposta por Rêgo (2014).
- Adicionar a análise de violações utilizando a ferramenta *PMD* à solução de um ambiente de *Data Warehousing-DWing* proposta por Rêgo (2014).
- Analisar um sistema adquirido por um Orgão Púlico utilizando à solução de um ambiente de *Data Warehousing-DWing* já contemplando das novas análises de bugs e violações.
- Construir *Dashboards* visando demonstrar de forma visual os dados chave das análises descrita no tópico anterior.

1.5 Metodologia de pesquisa

O que é uma pesquisa? De acordo com (GIL, 2002) é o procedimento racional e sistemático que tem como objetivo proporcionar respostas aos problemas que são propostos. A pesquisa é requerida quando não se dispõe de informação suficiente para responder ao problema, ou então quando a informação disponível se encontra em tal estado de desordem que não possa ser adequadamente relacionada ao problema.

Nessa seção, apresenta-se a metodologia de pesquisa adotada neste trabalho. Para isso, foram definidos: a natureza da pesquisa; o tipo de metodologia de pesquisa; o tipo de abordagem de pesquisa; os métodos de procedimentos de pesquisa e os tipos de técnicas de coletas de dados.

Os procedimentos de pesquisa selecionados foram pesquisa bibliográfica, documental, levantamento e estudo de caso. As técnicas de coleta de dados selecionadas foram entrevistas, questionários e registro de observação na vida real. A seleção metodológica é apresentada na Figura 21 e a descrição das classificações adotadas se encontra nas Tabelas 2 e 4.

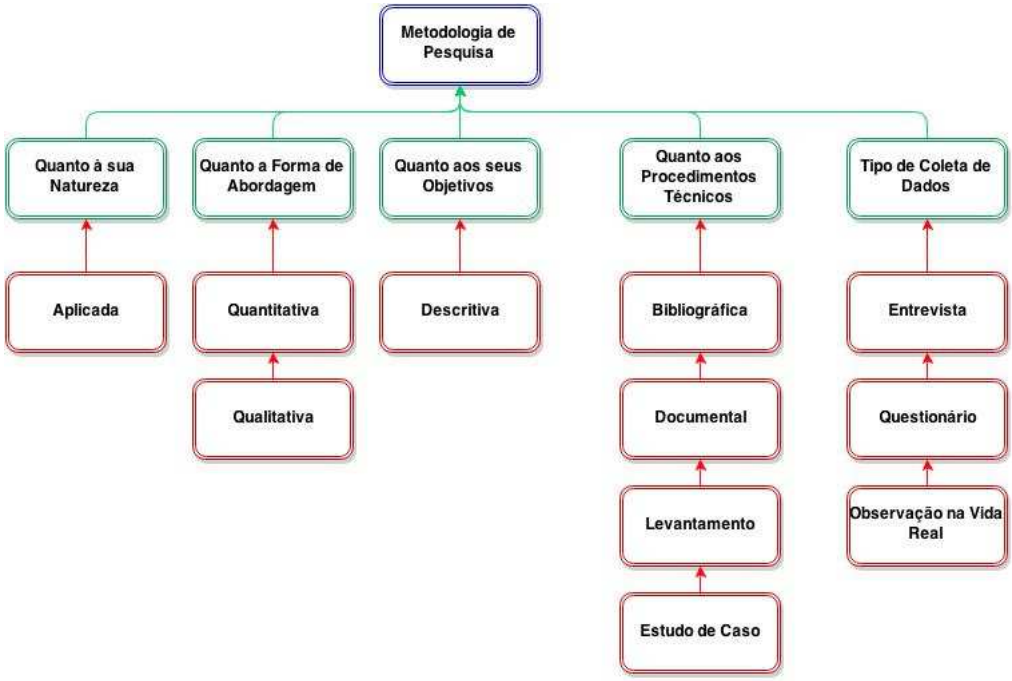

Figura 1 – Metodologia de Pesquisa

Ponto de Vista	**Selecionada**	**Descrição**
Natureza	Aplicada	Objetiva gerar conhecimentos para aplicação prática e dirigidos à solução de problemas específicos. Envolve verdades e interesses locais.
Forma de Abordagem	Quantitativa	Considera que tudo pode ser quantificável, o que significa traduzir em números opiniões e informações para classificá-las e analisá-las. Requer o uso de recursos e de técnicas estatísticas (percentagem, média, moda, mediana, desvio-padrão, coeficiente de correlação e análise de regressão).
	Qualitativa	Considera que há uma relação dinâmica entre o mundo real e o sujeito, isto é, um vínculo indissociável entre o mundo objetivo e a subjetividade do sujeito que não pode ser traduzido em números. A interpretação dos fenômenos e a atribuição de significados são básicas no processo de pesquisa qualitativa. O ambiente natural é a fonte direta para coleta de dados e o pesquisador é o instrumento-chave. Os pesquisadores tendem a analisar seus dados indutivamente
Objetivos	Descritiva	Visa descrever as características de determinada população ou fenômeno ou o estabelecimento de relações entre variáveis. Envolve o uso de técnicas padronizadas de coleta de dados: questionário e observação sistemática. Assume, em geral, a forma de Levantamento.
Procedimentos Técnicos	Pesquisa Bibliográfica	Quando elaborada a partir de material já publicado, constituído principalmente de livros, artigos de periódicos e atualmente com material disponibilizado na Internet.
	Pesquisa Documental	Quando elaborada a partir de materiais que não receberam tratamento analítico.
	Levantamento	Quando a pesquisa envolve a interrogação direta das pessoas cujo comportamento se deseja conhecer.
	Estudo de Caso	Quando envolve o estudo profundo e exaustivo de um ou poucos objetos de maneira que se permita o seu amplo e detalhado conhecimento.

Tabela 2 – Descrição das classificações adotadas de pesquisa, conceitos extraídos de Silva e Menezes (2005) parte 1.

Tipo de Coleta de Dados	Entrevista	É a obtenção de informações de um entrevistado, sobre determinado assunto ou problema.
	Questionário	É uma série ordenada de perguntas que devem ser respondidas por escrito pelo informante. O questionário deve ser objetivo, limitado em extensão e estar acompanhado de instruções As instruções devem esclarecer o propósito de sua aplicação, ressaltar a importância da colaboração do informante e facilitar o preenchimento.
	Observação na Vida Real	Quando se utilizam os sentidos na obtenção de dados de determinados aspectos da realidade. Na observação na vida real os registros de dados são feitos à medida que ocorrem.

Tabela 4 – Descrição das classificações adotadas de pesquisa, conceitos extraídos de Silva e Menezes (2005) parte 2.

Segundo (YIN, 2001), o estudo de caso é um conjunto de procedimentos pré-especificados para se obter uma investigação empírica que investiga um fenômeno contemporâneo dentro de seu contexto da vida real, especialmente quando os limites entre o fenômeno e o contexto não estão claramente definidos. Uma grande vantagem do estudo de caso é a sua capacidade de lidar com uma ampla variedade de evidências - documentos, artefatos, entrevistas e observações - além do que pode estar disponível no estudo histórico convencional. Além disso, em algumas situações, como na observação participante, pode ocorrer manipulação informal.

(WOHLIN et al., 2012) fraciona o estudo de caso em cinco passos. Nesta pesquisa, o estudo de caso compreende os passos: Planejar o Estudo de Caso; Coletar Dados; Analisar Dados Coletados e Compartilhar os Resultados. Portanto, os passos Projetar o Estudo de Caso e Preparar a Coleta de Dados definidas por Wohlin et al. (2012) foram agrupadas no passo Planejar o estudo de caso como na Figura 2.

O passo Planejar o Estudo de Caso consiste na determinação do objetivo e da questão de pesquisa, da escolha da metodologia de pesquisa, da definição das fases da pesquisa, da definição do procedimentos de pesquisa, do protocolo, das técnicas de coleta de dados e da proposta do trabalho final.

No passo Coletar Dados são executados os procedimentos de pesquisa e as técnicas de coletas de dados a seguir:

- Pesquisa Bibliográfica: pesquisa realizada a partir de livros, dissertações e trabalhos relacionados à área de pesquisa;

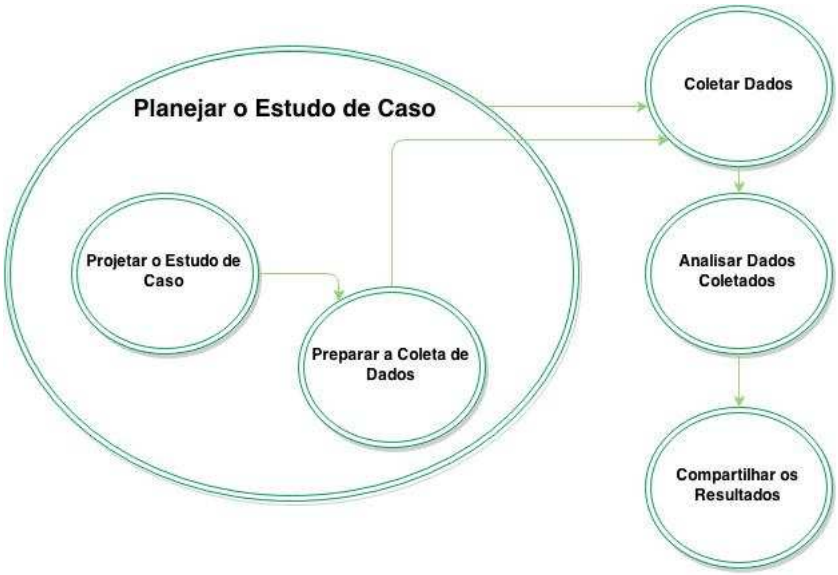

Figura 2 – Passos do Estudo de Caso

- Pesquisa Documental: pesquisa realizada a partir de documentos publicados por organizações públicas;
- Estudo de Caso: utilizar um estudo de caso real de uma organização pública brasileira;
- Entrevistas: dados serão coletados por meio de entrevistas, além de questionário, para incremento do estudo de caso;
- Documentos: coleta de dados dos documentos dos processos fornecidos pelo Órgãos público do estudo de caso será realizada para coleta de dados para análise;
- Observação na Vida Real: coleta de dados a partir da observação dentro do Órgão Público;

O passo Analizar Dados Coletados é onde os dados coletados serão analisados e interpretados. A análise compreende tanto a análise quantitativa quanto a análise qualitativa.

Por fim, o passo Compartilhar os resultados diz respeito expor os resultados de forma adequada para o leitor alvo.

1.6 Metodologia de Desenvolvimento

1.7 Organização do Trabalho

Esse trabalho está dividido em 5 capítulos:

- **Capítulo 2 - Métricas de Software:** Capítulo responsável pela explicação teórica a respeito do que são métricas de código e como elas foram utilizadas no desenvolvimento da solução que esse trabalho busca analisar.

- **Capítulo 3 - Contratações de Fornecedores de Desenvolvimento de Software:** Capítulo responsável por apresentar as principais informações referentes à Contratação de Fornecedores de Desenvolvimento de Software. Para isso, o capítulo é iniciado com uma visão geral sobre a importância das contratações e suas principais características. Posteriormente, é apresentado um resumo da Instrução Normativa 04.

- **Capítulo 4 - Data Warehouse:** Nesse capítulo, serão apresentados conceitos teóricos sobre *Data Warehousing*, assim como a maneira como foi desenvolvido e como funciona o ambiente de *Data Warehouse* para armazenamento de métricas de código fonte.

- **Capítulo 5 - Projeto de estudo de caso:** é apresentado o projeto do estudo de caso resultante do passo planejar Estudo de Caso, buscando demonstrar o escopo e elaborar um protocolo para o estudo de caso realizado. Elementos de pesquisa serão identificados e explicados como o problema a ser resolvido, os objetivos a serem alcançados no estudo de caso, quais os métodos de coleta e a análise dos dados.

- **Capítulo 6 - Conclusão:** Além das considerações finais dessa primeira parte do trabalho, serão descritos objetivos para o trabalho de conclusão de curso dois.

2 Métricas de Software

2.1 Processo de Medição

Segundo o SOFTEX (2013b), a Medição apoia a gerência e a melhoria de processo e de produto, sendo um dos principais meios para gerenciar as atividades do ciclo de vida do trabalho e avaliar a viabilidade dos planos de trabalho. O propósito da Medição é coletar e analisar os dados relativos aos produtos desenvolvidos e aos processos implementados na organização e em seus trabalhos, de forma a apoiar o efetivo gerenciamento e demonstrar objetivamente a qualidade dos produtos (ISO/IEC 15939, 2008).

Ainda segundo o SOFTEX (2013b), entende-se por método de medição uma sequência lógica de operações, descritas genericamente, usadas para quantificar um atributo com respeito a uma escala especificada. Esta escala pode ser nominal, ordinal ou de razão (de proporção), bem como definida em um intervalo:

- **Nominal:** A ordem não possui significado na interpretação dos valores (MEIRELLES, 2013)
- **Ordinal:** A ordem dos valores possui significado, porém a distância entre os valores não. (MEIRELLES, 2013)
- **Intervalo:** A ordem dos valores possui significado e a distância entre os valores também. Porém, a proporção entre os valores não necessariamente possui significado. (MEIRELLES, 2013)
- **Racional:** Semelhante a medida com escala do tipo intervalo, porém, a proporção possui significado. (MEIRELLES, 2013)

A ISO/IEC 15939 (2002) também divide o processo de medição em dois métodos diferentes, que se distinguem pela natureza do que é quantificado:

- **Subjetiva:** Quantificação envolvendo julgamento de um humano.
- **Objetiva:** Quantificação baseada em regras numéricas. Essas regras podem ser implementadas por um humano.

2.2 Definição das métricas de software

As métricas de software são medidas resultantes da medição do produto ou do processo do *software* pelo qual é desenvolvido, sendo que o produto de *software* deve ser visto como um objeto abstrato que se desenvolveu a partir de uma declaração inicial da

necessidade de um sistema para um software finalizado, incluindo o código-fonte e as várias formas de documentação produzidas durante o desenvolvimento (MILLS, 1999). Estas medidas resultantes podem ser estudas para serem utilizadas para medir a produtividade e a qualidade do produto.

Mills (1999) classifica as métricas de software como métricas de produtos ou métricas de processo, essa divisão pode ser vista na Figura 3.

- **Métricas de produtos:** são as medidas do produto de software em qualquer fase do seu desenvolvimento, a partir dos requisitos do sistema. Métricas de produto podem medir a complexidade da arquitetura do software, o tamanho do programa (código-fonte), ou o número de páginas de documentos produzidos.

- **Métricas de processo:** são as medidas do processo de desenvolvimento de software, como o tempo de desenvolvimento global, o tipo de metodologia utilizada, ou o nível médio da experiência da equipe de programação.

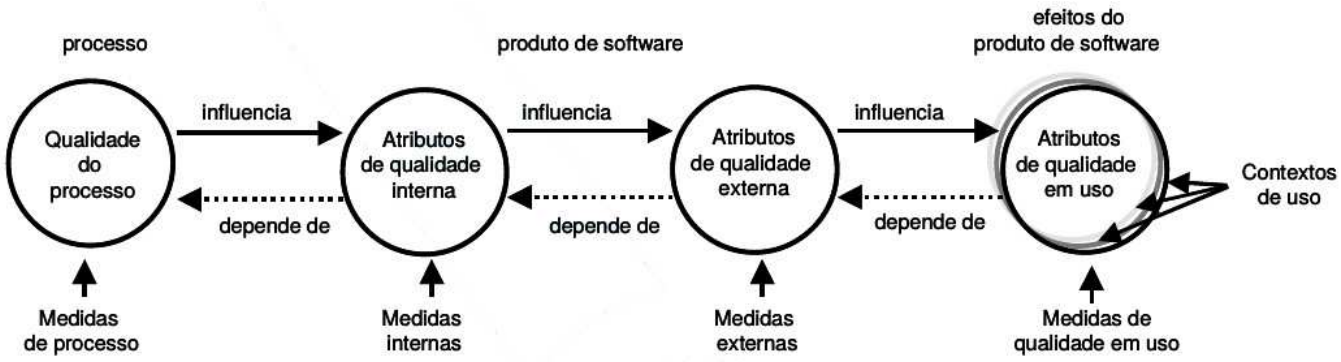

Figura 3 – Modelo de Qualidade do Produto da ISO/IEC 9126 (2001)

Na figura 3 também pode ser notada a classificação das métricas de acordo com os diferentes tipos de medição, refletindo como as métricas influenciam nos contextos em que elas estão envolvidas. A qualidade do produto de software pode ser avaliada medindo-se os atributos internos (tipicamente medidas estáticas de produtos intermediários), os atributos externos (tipicamente pela medição do comportamento do código quando executado) ou os atributos de qualidade em uso (ISO/IEC 9126, 2001).

- **Métricas internas:** Aplicadas em um produto de software não executável, como código fonte. Oferecem aos usuários, desenvolvedores ou avaliadores o benefício de poder avaliar a qualidade do produto antes que ele seja executável.

- **Métricas externas:** Aplicadas a um produto de software executável, medindo o comportamento do sistema do qual o software é uma parte através de teste, operação ou mesmo obervação. Oferecem aos usuários, desenvolvedores ou avaliadores o benefício de poder avaliar a qualidade do produto durante seu processo de teste ou operação.

- **Métricas de qualidade em uso:** Aplicadas para medir o quanto um produto atende as necessidades de um usuário para que sejam atingidas metas especificadas como eficácia, produtividade, segurança e satisfação.

2.3 Métricas de código fonte

A definição de código-fonte segundo a SCAM é qualquer descrição executável de um sistema de software sendo incluído código de máquina, linguagens de alto nível e por representações gráficas executáveis (HARMAN, 2010).

Segundo Mills (1999), a maior parte do trabalho inicial em métricas de produto analisou as características do código-fonte. A partir da experiência com métricas e modelos, tornou-se cada vez mais evidente que as informações de métricas obtidas anteriormente do ciclo de desenvolvimento pode ser de grande valor no controle do processo e dos resultados, o que sucedeu uma série de trabalhos tratando sobre o tamanho ou complexidade do software.

Neste capítulo, serão evidenciadas, em duas categorias as métricas de código-fonte que serão utilizadas neste trabalho de conclusão de curso, métricas de tamanho e complexidade e métricas de orientação a objetos. Estas métricas são objetivas e serão calculadas a partir da análise estática do código-fonte de um software.

2.3.1 Métricas de tamanho e complexidade

Cada produto do desenvolvimento de software é uma entidade física, como tal, pode ser descrito em termos de tamanho. Desde outros objetos físicos são facilmente mensuráveis (em comprimento, volume, massa, ou outra medida padrão), medir o tamanho do software deve ser relativamente simples e coerente de acordo com os princípios da teoria da medição. No entanto, na prática, a medição de tamanho apresenta grandes dificuldades (FENTON; PFLEEGER, 1998).

Segundo Honglei, Wei e Yanan (2009), as métricas de complexidade de *software* pertencem as principais medições de software e é o principal método para assegurar a qualidade do *software*. Quanto menor a complexidade dos programas, melhor eles são, assim, as métricas de complexidade também podem ser usadas para prever os defeitos ou erros. A seguir são apresentadas algumas métricas de tamanho e complexidade.

- **LOC** (*Lines of Code*): Métrica simples em que são contadas as linhas executáveis de um código, desconsiderando linhas em branco e comentários. (KAN, 2002)

- **ACCM** (*Average Cyclomatic Complexity per Method*): Mede a complexidade do programa, podendo ser representada através de um grafo de fluxo de controle. (MCCABE, 1976)

- **AMLOC** (*Average Method Lines of Code*): Indica a distribuição de código entre os métodos. Quanto maior o valor da métrica, mais pesado é o método. É preferível que haja muitos métodos com pequenas operações do que um método grande e de entendimento complexo. (MEIRELLES, 2013)

2.3.2 Métricas de Orientação a Objetos

Segundo Budd (2002), a programação orientada a objetos tornou-se extremamente popular nos últimos anos, inúmeros livros e edições especiais de revistas acadêmicas e comerciais têm surgido sobre o assunto. A julgar por essa atividade frenética, a programação orientada a objeto está sendo recebido com ainda mais entusiasmo do que vimos em ideias revolucionárias mais antigas, tais como programação estruturada "ou sistemas especialistas."

Há uma série de razões importantes pelas quais, nas últimas duas décadas, a programação orientada a objetos tornou-se o paradigma de programação dominante. A programação orientada a objeto possui uma ótima escala, desde o mais trivial dos problemas para a maioria das tarefas complexas. Ele fornece uma forma de abstração que reflete em técnicas de como pessoas usam para resolver problemas em sua vida cotidiana. E para a maioria das linguagens orientadas a objeto dominante há um número cada vez maior de bibliotecas que auxiliam no desenvolvimento de aplicações para muitos domínios (BUDD, 2002).

Serão adotadas neste trabalho as seguintes métricas de orientação a objetos:

- **ACC** (*Afferent Connections per Class* - Conexões Aferentes por Classe): Mede a conectividade entre as classes. Quanto maior a conectividade entre elas, maior o potencial de impacto que uma alteração pode gerar. (MEIRELLES, 2013)

- **ANPM** (*Average Number of Parameters per Method* - Média do Número de Parâmetros por Método): Indica a média de parâmetros que os métodos possuem. Um valor muito alto para quantidade de parâmetros pode indicar que o método está tendo mais de uma responsabilidade. (BASILI; ROMBACH, 1987)

- **CBO** (*Coupling Between Objects* - Acoplamento entre Objetos): Essa é uma métrica que diz respeito a quantas outras classes dependem de uma classe. É a conta das classes às quais uma classe está acoplada. Duas classes estão acopladas quando métodos de uma delas utilizam métodos ou variáveis de outra. Altos valores dessa métrica aumentam a complexidade e diminuem a manutenibilidade. (LAIRD, 2006).

- **DIT** (*Depth of Inheritance Tree* - Profundidade da Árvore de Herança): Responsável por medir quantas camadas de herança compõem uma determinada hierarquia de classes (LAIRD, 2006). Segundo Meirelles (2013), quanto maior o valor de DIT,

maior o número de métodos e atributos herdados, portanto, maior a complexidade.

- **LCOM4** (*Lack of Cohesion in Methods* - Falta de Coesão entre Métodos): A coesão de uma classe é indicada por quão próximas as variáveis locais estão relacionadas com variáveis de instância locais. Alta coesão indica uma boa subdivisão de classes. A LCOM mede a falta de coesão através dissimilaridade dos métodos de uma classe pelo emprego de variáveis de instância.(KAN, 2002). A métrica LCOM foi revista e passou a ser conhecida como LCOM4, sendo necessário para seu cálculo a construção de um gráfico não-orientado em que os nós são os atributos e métodos de uma classe. Para cada método deve haver uma aresta entre ele e outro método ou variável. O valor da LCOM4 é o número de componentes fracamente conectados a esse gráfico (MEIRELLES, 2013)

- **NOC** (*Number of Children* - Número de Filhos): É o número de sucessores imediatos, (portanto filhos) de uma classe. Segundo Laird (2006), altos valores indicam que a abstração da super classe foi diluída e uma reorganização da arquitetura deve ser considerada.

- **NOM** (*Number of Methods* - Número de Métodos): Indica a quantidade de métodos de uma classe, medindo seu tamanho. Classes com muitos métodos são mais difíceis de serem reutilizadas pois são propensas a serem menos coesas. (MEIRELLES, 2013)

- **NPA** (*Number of Public Attributes* - Número de Atributos Públicos): Mede o encapsulamento de uma classe, através da medição dos atributos públicos. O número ideal para essa métrica é zero (MEIRELLES, 2013)

- **RFC** (*Response For a Class* - Respostas para uma Classe): Kan (2002) define essa métrica como o número de métodos que podem ser executados em respostas a uma mensagem recebida por um objeto da classe.

2.4 Configurações de qualidade para métricas de código fonte

Meirelles (2013) apresentou uma abordagem para a observação das métricas de código-fonte em seu doutorado, onde estas métricas foram estudadas através de suas distribuições e associações. Foram avaliados a distribuição e correlações dos valores das métricas de 38 projetos de software livre, sendo coletados e analisados valores para cada métrica em mais de 344.872 classes e módulos. Segundo o próprio Meirelles (2013), entre as principais contribuições de sua tese foi a análise detalhada, em relação ao comportamento, valores e estudos de caso, de 15 métricas de código-fonte.

Dentre os objetivos científicos da tese de Meirelles (2013) o mais crucial para este trabalho de conclusão de curso é o objetivo OC4, que trata das distribuições estatísticas

dos valores de métricas em 38 projetos de software livre, a fim de compreender qual a abordagem estatística mais informativa para o monitoramento dessas métricas, bem como observar os valores frequentes para essas métricas, de forma a servirem de referência para projetos futuros.

Meirelles (2013), para conduzir o seu estudo quantitativo, utilizou-se da técnica de estatística descritiva: percentil. O percentil separa os dados em cem grupos que apresentam o mesmo número de valores, sendo a definição do percentil de ordem px100(0<p<1), em um conjunto de dados de tamanho n, é o valor da variável que ocupa a posição px(n+1) do conjunto de dados ordenados. O percentil de ordem p (ou p-quantil) deixa px100% das observações abaixo dele na amostra ordenada. O percentil 25 recebe o nome de primeiro quartil, o percentil 50 de segundo quartil ou mediana, e o percentil 75 de terceiro quartil. Uma das hipóteses que Meirelles (2013) utiliza é que só a partir do terceiro quartil será possível obter dados informativos sobre as métricas.

Após a aplicação da técnica percentil na série de dados das métricas de código-fonte obtidos da análise estática do código-fonte dos projetos de software livre, tabelas apresentando os valores percentis de cada métrica nos 38 projetos avaliados foram criadas, como a Tabela 5, que mostra os percentis para a métrica NOC.

	Mín	**1%**	**5%**	**10%**	**25%**	**50%**	**75%**	**90%**	**95%**	**99%**	**Máx**
Eclipse	0,0	0,0	0,0	0,0	0,0	0,0	0,0	2,0	3,0	10,0	243,0
Open JDK8	0,0	0,0	0,0	0,0	0,0	0,0	0,0	1,0	3,0	9,0	301,0
Ant	0,0	0,0	0,0	0,0	0,0	0,0	0,0	1,0	3,0	12,0	162,0
Checkstyle	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	2,0	5,0	144,0
Eclipse Metrics	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	2,0	11,0	24,0
Findbugs	0,0	0,0	0,0	0,0	0,0	0,0	0,0	1,0	2,0	5,0	55,0
GWT	0,0	0,0	0,0	0,0	0,0	0,0	0,0	39,0	1,0	8,0	398,0
Hudson	0,0	0,0	0,0	0,0	0,0	0,0	0,0	1,0	2,0	7,0	23,0
JBoss	0,0	0,0	0,0	0,0	0,0	0,0	0,0	1,0	2,0	6,0	256,0
Kalibro	0,0	0,0	0,0	0,0	0,0	0,0	0,0	1,0	2,0	9,0	101,0
Log4J	0,0	0,0	0,0	0,0	0,0	0,0	0,0	1,0	3,0	9,0	23,0
Netbeans	0,0	0,0	0,0	0,0	0,0	0,0	0,0	1,0	2,0	6,0	989,0
Spring	0,0	0,0	0,0	0,0	0,0	0,0	0,0	2,0	3,0	7,0	175,0
Tomcat	0,0	0,0	0,0	0,0	0,0	0,0	0,0	1,0	2,0	8,0	82,0

Tabela 5 – Percentis para métrica NOC em projetos Java extraídos de Meirelles (2013)

Através dos resultados obtidos para cada métrica, Meirelles (2013) observou que era possível identificar valores frequentes analisando os percentis. Na tabela 5, por exemplo, considerando o projeto **Open JDK8** como referência, foram observados os intervalos de valores de 0, como muito frequente, 1 e 2 como frequente, 3 como pouco frequente e acima de 3 como não frequente (MEIRELLES, 2013).

Rêgo (2014), observando o trabalho de análise de percentis de Meirelles (2013),

percebeu que é possível utilizar os intervalos de frequência obtidos como uma evidência empírica de qualidade do código-fonte. Rêgo (2014) também renomeou os intervalos de frequência obtidos por Meirelles (2013), como na Tabela 6, a fim de facilitar a interpretação de métricas de código-fonte.

Intervalo de Frequência	Intervalo Qualitativo
Muito Frequente	Excelente
Frequente	Bom
Pouco Frequente	Regular
Não Frequente	Ruim

Tabela 6 – Nome dos Intervalos de Frequência extraídos de Rêgo (2014)

Rêgo (2014) observando a diferença dos valores das métricas entre os projetos analisados e na tentativa de diminuir tamanha diferença, considerou dois cenários. Foram utilizado dois produtos de *software* como referências para cada uma das métricas na linguagem de programação Java. Em um primeiro cenário, foi analisado o *Open JDK8*, software que contia os menores valores percentis para as métricas. Já em um segundo cenário, foi considerado o Tomcat, contendo os valores percentis mais altos. A tabela 7 mostra o resultado desta análise:

Métrica	Intervalo Qualitativo	OpenJDK8 Metrics	Tomcat Metrics
LOC	Excelente	[de 0 a 33]	[de 0 a 33]
	Bom	[de 34 a 87]	[de 34 a 105]
	Regular	[de 88 a 200]	[de 106 a 276]
	Ruim	[acima de 200]	[acima de 276]
ACCM	Excelente	[de 0 a 2,8]	[de 0 a 3]
	Bom	[de 2,9 a 4,4]	[de 3,1 a 4,0]
	Regular	[de 4,5 a 6,0]	[de 4,1 a 6,0]
	Ruim	[acima de 6]	[acima de 6]
AMLOC	Excelente	[de 0 a 8,3]	[de 0 a 8]
	Bom	[de 8,4 a 18]	[de 8,1 a 16,0]
	Regular	[de 19 a 34]	[de 16,1 a 27]
	Ruim	[acima de 34]	[acima de 27]
ACC	Excelente	[de 0 a 1]	[de 0 a 1,0]
	Bom	[de 1,1 a 5]	[de 1,1 a 5,0]
	Regular	[de 5,1 a 12]	[de 5,1 a 13]
	Ruim	[acima de 12]	[acima de 13]
ANPM	Excelente	[de 0 a 1,5]	[de 0 a 2,0]
	Bom	[de 1,6 a 2,3]	[de 2,1 a 3,0]
	Regular	[de 2,4 a 3,0]	[de 3,1 a 5,0]
	Ruim	[acima de 3]	[acima de 5]
CBO	Excelente	[de 0 a 3]	[de 0 a 2]
	Bom	[de 4 a 6]	[de 3 a 5]
	Regular	[de 7 a 9]	[de 5 a 7]
	Ruim	[acima de 9]	[acima de 7]
DIT	Excelente	[de 0 a 2]	[de 0 a 1]
	Bom	[de 3 a 4]	[de 2 a 3]
	Regular	[de 5 a 6]	[de 3 a 4]
	Ruim	[acima de 6]	[acima de 4]
LCOM4	Excelente	[de 0 a 3]	[de 0 a 3]
	Bom	[de 4 a 7]	[de 4 a 7]
	Regular	[de 8 a 12]	[de 8 a 11]
	Ruim	[acima de 12]	[acima de 11]
NOC	Excelente	[0]	[1]
	Bom	[1 a 2]	[1 a 2]
	Regular	[3]	[3]
	Ruim	[acima de 3]	[acima de 3]
NOM	Excelente	[de 0 a 8]	[de 0 a 10]
	Bom	[de 9 a 17]	[de 11 a 21]
	Regular	[de 18 a 27]	[de 22 a 35]
	Ruim	[acima de 27]	[acima de 35]
NPA	Excelente	[0]	[0]
	Bom	[1]	[1]
	Regular	[de 2 a 3]	[de 2 a 3]
	Ruim	[acima de 3]	[acima de 3]
RFC	Excelente	[de 0 a 9]	[de 0 a 11]
	Bom	[de 10 a 26]	[de 12 a 30]
	Regular	[de 27 a 59]	[de 31 a 74]
	Ruim	[acima de 59]	[acima de 74]

Tabela 7 – Configurações para os Intervalos das Métricas para Java extraídas de Rêgo (2014)

2.5 Cenários de limpeza

Segundo Marinescu (2005), a utilização de métricas isoladas dificulta a interpretação de anomalias do código, reduzindo a aplicabilidade da medição feita. Além disso, o autor ainda afirma que a métrica por si só não contém informação suficiente para motivar uma transformação no código que melhore sua qualidade. Uma das soluções para sanar o problema da interpretação do código através de métricas isoladas é interpretar o código através de cenários de limpeza.

Machini et al. (2010) apresenta uma maneira de interpretar os valores das métricas através de cenários problemáticos e suas possíveis melhorias, para que as métricas possam ser mais facilmente incorporadas no cotidiano dos programadores. Machini et al. (2010) também apresenta um estilo de programação baseado no paradigma da Orientação a Objetos que busca o que denominamos de "Código Limpo", concebido e aplicado por renomados desenvolvedores de software como Robert C. Martin (MARTIN, 2008) e Kent Beck (BECK, 2007)

Segundo Beck (2007), um código limpo está inserido em um estilo de programação que busca a proximidade a três valores: expressividade, simplicidade e flexibilidade.

- **Expressividade:** Um código se expressa bem quando alguém que o lê é capaz de compreendê-lo e modificá-lo. Beck (2007) destaca que quando foi necessário modificar um código, ele gastou muito mais tempo lendo o que já havia sido feito do que escrevendo sua modificação.

- **Simplicidade:** Um código é simples quando pode ser facilmente lido, havendo uma redução da quantidade de informação que o leitor deve compreender para fazer alterações. Eliminar o excesso de complexidade faz com que aqueles que estejam lendo o código o entenda mais rapidamente.

- **Flexibilidade:** Capacidade de estender a aplicação alterando o mínimo possível a estrutura já criada.

Machini et al. (2010) apresenta em seu trabalho diversas técnicas para a obtenção de um código limpo. Um resumo destas técnicas são apresentadas nas tabelas 8, 9, 10 e 11.

Técnicas de Limpeza	Descrição	Contribuições	Consequências
Composição de Métodos	Compor os métodos em chamadas para outros rigorosamente no mesmo nível de abstração.	• Facilidade de entendimento de métodos menores • Criação de métodos menores com nomes explicativos	• Menos Operações por Método • Mais Parâmetros de Classe • Mais Métodos na Classe
Métodos Explicativos	Criar um método que encapsule uma operação pouco clara, geralmente associada a um comentário	• O código cliente do método novo terá uma operação com nome que melhor se encaixa no contexto.	• Mais métodos na classe.
Métodos como Condicionais	Criar um método que encapsule uma expressão booleana para obter condicionais mais claras.	• Facilidade na leitura de condicionais no código cliente. • Encapsulamento de uma expressão booleana.	• Mais métodos na classe.
Evitar Estruturas Encadeadas	Utilizar a composição de métodos para minimizar a quantidade de estruturas encadeadas em cada método (if, else).	• Facilidade para a criação de testes. • Cada método terá estruturas mais simples e fáceis de serem compreendidas.	• Menos Estruturas encadeadas por método (if e else) • Benefícios do Uso de Composição de Métodos

Tabela 8 – Conceitos de Limpeza levantados por Machini et al. (2010) e adaptados por Rêgo (2014) parte 1.

Técnicas de Limpeza	Descrição	Contribuições	Consequências
Cláusulas Guarda	Criar um retorno logo no inicio de um método ao invés da criação de estruturas encadeadas com if sem else.	• Estruturas de condicionais mais simples. • Leitor não espera por uma contrapartida do condicional(ex:if sem else).	• Menos estruturas encadeadas na classe.
Objeto Método	Criar uma classe que encapsule uma operação complexa simplificando a original(cliente).	• O código cliente terá um método bastante simples. • Nova classe poderá ser refatorada sem preocupações com alterações no código cliente. • Nova classe poderá ter testes separados.	• Menos operações no método cliente. • Menos responsabilidades da classe cliente. • Mais classes. • Mais acoplamento da classe cliente com a nova classe.
Evitar *Flags* como Argumentos	Ao invés de criar um método que recebe uma flag e tem diversos comportamentos, criar um método para cada comportamento.	• Leitor não precisará entender um método com muitos condicionais. • Testes de unidade independentes.	• Mais métodos na classe.

Tabela 9 – Conceitos de Limpeza levantados por Machini et al. (2010) e adaptados por Rêgo (2014) parte 2.

Técnicas de Limpeza	Descrição	Contribuições	Consequências
Maximizar a Coesão	Quebrar uma classe que não segue o Princípio da Responsabilidade Única: as classes devem ter uma única responsabilidade, ou seja, ter uma única razão para mudar.	• Cada classe terá uma única responsabilidade. • Cada classe terá seus testes independentes. • Sem interferências na implementação das responsabilidades.	• Mais Classes • Menos Métodos em cada Classe • Menos Atributos em cada Classe
Objeto como Parâmetro	Localizar parâmetros que formam uma unidade e criar uma classe que os encapsule.	• Menor número de parâmetros facilita testes e legibilidade. • Criação de uma classe que poderá ser reutilizada em outras partes do sistema.	• Menos Parâmetros sendo passados para Métodos • Mais Classes
Parâmetros como Variável de Instância	Localizar parâmetro muito utilizado pelos métodos de uma classe e transformá-lo em variável de instância.	• Não haverá a necessidade de passar longas listas de parâmetro através de todos os métodos.	• Menos Parâmetros passados pela Classe • Possível diminuição na coesão
Uso de Exceções	Criar um fluxo normal separado do fluxo de tratamento de erros utilizando exceções ao invés de valores de retornos e condicionais.	• Clareza do fluxo normal sem tratamento de erros através de valores de retornos e condicionais.	• Menos Estruturas encadeadas.

Tabela 10 – Conceitos de Limpeza levantados por Machini et al. (2010) e adaptados por Rêgo (2014) parte 3.

Técnicas de Limpeza	Descrição	Contribuições	Consequências
Delegação de Tarefa	Transferir um método que utiliza dados de uma classe "B"para a "B".	• Redução do acoplamento entre classes. • Proximidade dos métodos e dados sobre os quais trabalham.	• Menos métodos na classe inicial. • Mais métodos na classe que recebe o novo método.
Objeto Centralizador	Criar uma classe que encapsule uma operação com alta dependência entre classes.	• Simplificação da classe cliente. • Redução do acoplamento da classe cliente com as demais. • Nova classe poderá receber testes e melhorias independentes.	• Menos operações no método cliente. • Menos responsabilidades da classe cliente. • Mais classes. • Mais acoplamento da classe cliente com a nova classe.
Uso Excessivo de Herança	Localizar uso excessivo de herança e transformá-lo em agregação simples.	• Redução do acoplamento entre as classes.	• Maior Flexibilidade de Adição de Novas Classes. • Menor Acoplamento entre as classes.
Exposição Pública Excessiva	Localizar uso excessivo de parâmetros públicos e transformá-lo em parâmetros privados.	• Menor Acoplamento entre as classes.	• Maior Encapsulamento de Parâmetros.

Tabela 11 – Conceitos de Limpeza levantados por Machini et al. (2010) e adaptados por Rêgo (2014) parte 4.

Após apresentar as técnicas para obtenção de um código limpo, Machini et al. (2010) adota uma abordagem baseada em cenários para identificar trechos de código com características indesejáveis. Os cenários devem possuir um contexto criado a partir de

poucos conceitos de código limpo no qual um pequeno conjunto de métricas é analisado e interpretado através da combinação de seus valores. A ideia principal desta abordagem é facilitar melhorias de implementação e a procura por problemas quanto a limpeza de código através da aproximação dos valores das métricas com os esperados nos contextos de interpretação (MACHINI et al., 2010).

Rêgo (2014) utiliza a mesma abordagem de cenários que Machini et al. (2010), onde algumas técnicas de limpeza de código apresentadas nas tabelas 8, 9, 10 e 11 são correlacionadas com as métricas da Seção 2.3. Adicionalmente, utiliza-se a configuração do *Open JDK8* considerando como valores altos os valores obtidos pelos intervalos Regular e Ruim tal como mostrados na Tabela 7.

Aproveitando inicialmente os cenários **Classe pouco coesa** e **Interface dos métodos** extraídos de Machini et al. (2010), Rêgo (2014) elaborou mais alguns cenários de limpeza. O resultado dessa atividade pode ser visto na tabela 12:

Cenário de Limpeza	Conceito de Limpeza	Características	Recomendações	Forma de Detecção pelas Métricas de Código-Fonte	Padrões de Projeto Associados
Classe Pouco Coesa	Maximização da Coesão	Classe Subdivida em grupos de métodos que não se relacionam	Reduzir a subdvisão da Classe	Intervalos Regulares e Ruins de LCOM4, RFC.	*Chain of Reponsabilities, Mediator, Decorator.*
Interface dos Métodos	Objetos como Parâmetro e Parâmetros como Variáveis de Instância	Elevada Média de parâmetros repassados pela Classe	Minimizar o número de Parâmetros.	Intervalos Regulares e Ruins de ANPM.	*Facade, Template Method, Strategy, Command, Mediator, Bridge.*
Classes com muitos filhos	Evitar Uso Excessivo de Herança	Muitas Sobreescritas de Métodos	Trocar a Herança por uma Agregação.	Intervalos Regulares e Ruins de NOC.	*Composite, Prototype, Decorator, Adapter.*
Classe com métodos grandes e/ou muitos condicionais	Composição de Métodos, Evitar Estruturas Encadeadas Complexas	Grande Número Efetivo de Linhas de Código	Reduzir LOC da Classe e de seus métodos, Reduzir a Complexidade Ciclomática e Quebrar os métodos.	Intervalos Regulares e Ruins de AMLOC, ACCM.	*Chain of Reponsabilities, Mediator, Flyweight* .
Classe com muita Exposição	Parâmetros Privados	Grande Número de Parâmetros Públicos	Reduzir o Número de Parâmetros Públicos.	Intervalos Regulares e Ruins de NPA.	*Facade, Singleton.*
Complexidade Estrutural	Maximização da Coesão	Grande Acoplamento entre Objetos	Reduzir a a quantidade de responsabilidades dos Métodos.	Intervalos Regulares e Ruins de CBO e LCOM4.	*Chain of Responsabilities, Mediator, Strategy.*

Tabela 12 – Cenários de Limpeza extraídos de Rêgo (2014)

2.6 Considerações Finais do Capítulo

Esse capítulo apresentou a fundamentação teórica sobre métricas de código fonte, quais são seus intervalos qualitativos e também foi apresentada a maneira como elas foram relacionadas a cenários de limpeza. O próximo capítulo será responsável por apresentar a importância dos Verificadores Estáticos de Código.

3 Verificadores Estáticos de Código

3.1 Verificação de Código

Os verificadores estáticos de código, são ferramentas automáticas para a verificação de código, que podem verificar estilos de programação, erros ou ambos. O verificador de erro é uma ferramenta de análise estática voltada para a detecção automática de erros, tomando como base as tendências comuns dos desenvolvedores em atrair defeitos que, em sua maioria, não são visíveis aos compiladores. (LOURIDAS, 2006)

Segundo Pugh (2008), análise Estática de Código é a análise de um sistema de computador que é realizada sem a sua execução, já a análise realizada com a execução dos programas é conhecida como análise dinâmica. O termo Análise Estática de Código pode se referir à análise automatizada, que é uma das técnicas estáticas de inspeção de software no processo de validação e verificação de software. A Inspeção é uma forma de análise estática na qual você examina o programa sem executá-lo que identificam erros comuns em diferentes linguagens de programação. A possibilidade de automatizar o processo de verificação de programas resultou no desenvolvimento de analisadores estáticos automatizados.

Analisadores estáticos automatizados são ferramentas de software que varrem o texto-fonte, entretanto os verificadores estáticos de código podem trabalhar diretamente sobre o código-fonte do programa ou trabalhar sobre o código objeto, no caso da linguagem Java, sobre o *bytecode* (SOMMERVILLE et al., 2008). Segundo Louridas (2006), a intenção da análise estática automatizada é chamar a atenção para anomalias do programa. As anomalias são resultados de erros de programação ou omissões, de tal modo que onde possa ocasionar erros durante a execução do programa é enfatizado, porém nem todas as anomalias são necessariamente defeitos de programa.

Jelliffe (2004), destacou alguns dos benefícios da utilização de analisadores estáticos:

- Encontra erros e códigos de risco;
- Fornece um retorno objetivo aos programadores para ajudá-los a reconhecer onde eles foram precisos ou desatentos;
- Fornece a um líder de projeto uma oportunidade para estudar o código, o projeto e a equipe de uma perspectiva diferente;
- Retira certas classes de defeitos, o que possibilita que a equipe concentre-se mais

nas deficiências do projeto.

3.2 Ferramentas de verificação de Erro

Neste trabalho foram utilizadas duas ferramentas de verificação de erros, o FindBugs na versão 3.0.1 e o PMD na versão 5.0.0.

O FindBugs é uma ferramenta de código aberto utilizado pelos desenvolvedores de software para fazer uma inspeção no código de forma automatizada. Esta ferramenta examina as sua classes procurando por possíveis erros em potencial no código durante a fase de desenvolvimento. O FindBugs analisa o código fonte ou mesmo o código objeto, *bytecode* para programas Java. Segundo Louridas (2006), é um popular verificador estático de código para a linguagem Java e considera a ferramenta como um patrocinador na fortificação da qualidade do software. Atualmente, ela pode analisar programas compilados em qualquer versão da linguagem Java.

O FindBugs trabalha, basicamente, com as seguintes categorias de bug:

- Más práticas (*Bad practice*)
- Corretude (*Correctness*) f
- Erros de concorrência (*Multithreaded correctness*)
- Internacionalização (*Internationalization*)
- Experimental (*Experimental*)
- Vulnerabilidade de código malicioso (*Malicious code vulnerability*)
- Potenciais problemas de desempenho (*Performance*)
- Segurança (*Security*)
- Código Confuso (*Dodgy code*)

O PMD é um analisador de código Java que procura em uma base de código por possíveis problemas voltados às más práticas de desenvolvimento, tais como variáveis não utilizadas, código duplicado, trechos de código de elevada complexidade, criação desnecessária de objetos. Segundo (HOVEMEYER; PUGH, 2004) é uma ferramenta de extrema valia em forçar os desenvolvedores a seguir um estilo de programação e em tornar o código mais fácil para ser entendido pelo desenvolvedores.

O PMD trabalha, basicamente, com os seguintes conjunto de regras para linguagem java:

- *Basic rules* - Regras básicas gerais.

- *Braces rules* - Regras relacionadas ao uso de chaves.
- *Code size rules* - Regras que avaliam questões relacionadas ao tamanho do código.
- *Controversial rules* - Regras de aplicação geral, mas de aplicação controversa.
- *Coupling rules* - Regras relacionadas ao acoplamento entre objetos e pacotes.
- *Design rules* - Regras que avaliam o design do código.
- *Import statement rules* - Regras relacionadas ao uso de import.
- *Naming rules* - Regras que avaliam nomes de variáveis e métodos.
- *Optimization rules* - Regras relacionadas à otimização.
- *Strict Exception rules* - Regras relacionadas ao lançamento e captura de exceções.
- *String and StringBuffer rules* - Regras que verificam o bom uso das classes *String* e *StringBuffer*.
- *Unused code rules* - Regras que detectam código não utilizado.

3.3 Considerações Finais do Capítulo

Nesse capítulo foi apresentado as ferramentas de verificação de erro utilizadas neste trabalho, bem como a base teórica para sua compreensão. No próximo capítulo será apresentado a importância da utilização de um *dashboard*.

4 Dashboard

4.1 A Importância do *Dashboard*

A orientação visual dos *dashboards* é importante devido à velocidade da percepção de que é geralmente necessária para monitorar informações. Quanto mais rápido se deseja avaliar o que está acontecendo, mais deve-se confiar no meio gráfico para mostrar a informação. O texto deve ser lido, o que envolve um processo relativamente lento, sendo que certas propriedades visuais, no entanto, pode ser percebida de relance, sem pensamento consciente.

O processo de monitorização visual envolve uma série de passos sequenciais que o *dashboard* deve ser concebido para apoio. O usuário ao começar a obter uma visão geral do que está acontecendo deve rapidamente identificar o que precisa de atenção, para que em seguida, o usuário olhe mais de perto cada uma dessas áreas que precisam de atenção para entendê-las bem o suficiente para determinar se algo deve ser feito sobre eles. O monitoramento é uma atividade cognitiva que recebe a entrada principalmente através do canal visual, porque este é o sentido mais poderoso, que é capaz de trabalhar em altas velocidades de entrada, capaz de detectar diferenças sutis e complexas.(FEW, 2006)

Segundo Few (2006), um *dashboard* é um display visual das informações mais importantes necessárias para alcançar um ou mais objetivos, consolidados e organizados em uma única tela para que a informação possa ser monitorada em um piscar de olhos. Few (2006) categorizou diversos tipos de *dashboards* sendo estratégicas, analíticas, ou operacionais, e as características do design no que tange à sugestão de organização variam para dar suporte às necessidades de cada categoria:

- fins estratégicos - O uso primários de *dashboards* nos dias de hoje é para propósitos estratégicos, oferecem uma rápida visão que os tomadores de decisão precisam para monitorar a saúde e as oportunidades de um negócio.

- fins analíticos - Mais sofisticação para as mídias de exibição, para que os analistas possam examinar melhor dados complexos e relacionamentos. *Dashboards* analíticos devem suportar interações com os dados, como aprofundamentos em camadas detalhadas, não apenas para ver o que está acontecendo, mas para examinar as causas.

- fins operacionais - *Dashboards* que monitorem operações devem manter consciência das atividades e eventos que estão mudando constantemente e podem demandar atenção e resposta.

Além de categorizar os tipos de *dashboards*, Few (2006) também evidencia o primordial para se obter um *dashboard* de qualidade:

- Disponibilizar a informação diretamente relacionada num único ecrã, ou seja, evitar partir a informação por várias páginas;
- Evitar a necessidade de *scrolling*;
- Contextualizar a informação disponibilizada;
- Incluir fatores de comparação e sugerir ações na visualização dos indicadores;
- Utilizar escalas adequadas, que devem dar uma perspectiva real das quantidades apresentadas e não podem iludir os utilizadores;
- Utilizar níveis de precisão adequados nos indicadores, pois evita perdas de tempo com leituras e interpretações de informação desnecessárias e pouco relevantes;
- Escolher os indicadores mais adequados, que facilitem e acelerem a interpretação da informação disponibilizada;
- Escolher soluções gráficas flexíveis e adequadas que facilitem e acelerem a interpretação da informação disponibilizada;
- Uniformizar a leitura ao longo do *dashboard*;
- Facilitar a interpretação da informação disponível para acelerar a sua leitura. Por exemplo, evitar cores berrantes, muito próximas, muito apagadas ou um número muito elevado de cores;
- Apresentar a informação de forma equilibrada, dado que o espaço utilizado num *dashboard* desce de importância do canto superior esquerdo para o canto inferior direito, e por esta razão, a informação que se destaca na visualização deverá ser a mais importante;
- Os títulos não devem ser mais apelativos que os indicadores;
- Destacar a informação mais importante e não cair no erro de chamar a atenção para tudo;
- Aproveitar bem o espaço disponível, ou seja evitar decorações desnecessárias e ainda evitar soluções de pesada implementação para responder a pormenores visuais;
- Utilizar cores de forma ponderada, ou seja, utilizar cores apelativas apenas para a informação mais importante, podendo utilizar contrastes;
- Manter as cores para os mesmos indicadores ao longo do *dashboard* ou para o mesmo

tipo de indicador associado;

- Podem ser utilizadas figuras geométricas para além das cores, tais como o círculo, triângulo ou quadrado como forma de ajudar utilizadores que sofram de daltonismo;
- Criar uma apresentação apelativa, baseando-se no nossa intuição e naquilo que consideramos que a maioria das pessoas aceita e tolera positivamente.

4.2 Ferramenta para criação de *Dashboards*

Neste trabalho foi utilizado o CDE (*community dashboard editor*) na versão 14.12.10.1 para a criação dos *dashboards*. O CDE é um *plugin* para o *Pentaho Business Intelligence*.

O CDE permite o desenvolvimento e a implantação de *dashboards* no *Pentaho Business Intelligence*. O CDE nasceu para simplificar a criação e edição de *dashboards* e é uma ferramenta muito poderosa e completa, combinando *front end* com fontes de dados e componentes personalizados de uma forma perfeita. (CDE. . . , 2014)

O CDE possui algumas tecnologia por trás dela:

- CDF (*Community Dashboard Framework*) - É um *framework* HTML/javascript que permite criar páginas com relatórios, gráficos e tabelas.
- CDA (*Community Dashboard Access*) - São vários componentes que dão acesso à diferentes tipos de fontes de dados.
- CCC (*Community Chart Components*) - É uma biblioteca de gráficos, que possui um poderoso conjunto de ferramentas de visualização livre e de código aberto. O objetivo do CCC é fornecer aos desenvolvedores o caminho para incluir em seus *dashboards* os tipos de gráficos básicos, sem perder o princípio fundamental: a extensibilidade.

Para o design do *dashboard*, o CDE oferece três perspectivas:

- *Layout* - Para projetar o layout do seu dashboard a partir do zero ou de um templete, ao definir o layout é possível aplicar estilos e adicionar elementos HTML para descrever páginas Web, CSS para controlar o estilo do layout, *JavaScript* para adicionar interatividade e *jQuery* para simplificar todas essas tarefas.
- *Components* - Para adicionar e configurar os diferentes componentes que compõem o seu *dashboard*: Componentes Visuais (caixas de texto, tabelas e gráficos, seletores e relatórios), parâmetros que representam os valores que são partilhados pelos componentes e *Scripts*, que permitem personalizar a aparência ou o comportamento de outros componentes.
- *Datasources* - Define os dados usados pelos componentes. *Dashboards* podem ser

povoados por uma variedade de fontes como: Bases de dados, cubos do *Mondrian*, metadados do Pentaho , arquivos XML, *ad-hoc datasource* e transformações do *Kettle*.

4.3 Considerações Finais do Capítulo

Nesse capítulo foi apresentado as ferramentas para a criação de *Dashboards* utilizadas neste trabalho, bem como a sua importância. O próximo capítulo será responsável por apresentar a importância das contratações e o resumo da Instrução Normativa 04.

5 Contratações de Fornecedores de Desenvolvimento de Software

O não cumprimento do disposto na legislação de licitações e contratos pode acarretar riscos para a contratação de Tecnologia da Informação, sendo de grande importância o conhecimento da legislação de licitações e contratos para que se possam ser usados como base em processos de contratação de fornecedores de desenvolvimento de software. Assim, neste capítulo será apresentada uma visão sobre a importância da contratação de serviços de TI e de forma resumida, os conceitos de contratação de serviços de TI presentes na Instrução Normativa Nº 04.

5.1 Importância da Contratação de Fornecedores de Desenvolvimento de Software

De acordo com a ISO/IEC 15939 (2008), contrato é o cordo realizado entre duas partes, respaldado pela lei, ou acordo interno similar restrito a uma organização, para o fornecimento de serviços de software ou para o fornecimento, desenvolvimento, produção, operação ou manutenção de um produto de software.

Embora a contratação de serviços de TI tenha papel importante na estratégia organizacional, há muitos riscos que podem frustrar seus resultados. A definição e institucionalização de processos de contratação de serviços de TI, especialmente aqueles relacionados a software, envolvem ações complexas, principalmente no que diz respeito à identificação dos requisitos necessários, a garantia da qualidade dos resultados esperados, os critérios de aceitação, a gestão de mudanças, as transferências de conhecimentos, a legislação pertinente, entre outros. E envolvem também questões de relacionamento entre clientes e fornecedores, o que implica em competências administrativas e jurídicas. Essas complexidades apresentam riscos para as partes envolvidas e, como consequência, é comum a ocorrência de sérios conflitos. Normas e modelos de referência podem ser úteis para resolver esses conflitos (CRUZ; ANDRADE; FIGUEIREDO, 2011).

Neste trabalho, a aquisição de *software* é o assunto mais relevante dentro do contexto de contratos, mais especificamente da atividade de aceitação pelo cliente. A (ISO/IEC 15939, 2008) dividi a aquisição de *software* e serviços correlatos em quatro atividades, conforme a Figura 4.

Segundo a ISO/IEC 15939 (2008), o propósito da atividade de aceitação pelo cliente é aprovar o *software* e os serviços correlatos (S&SC) entregues pelo fornecedor

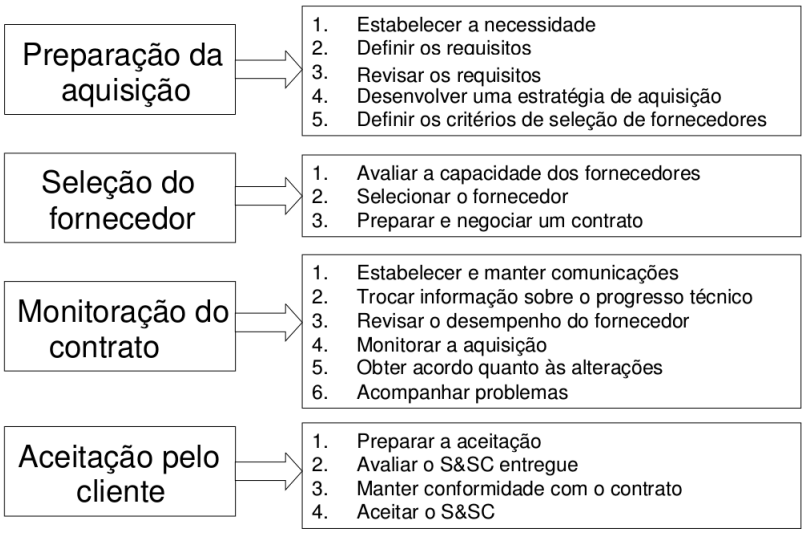

Figura 4 – Atividades de aquisição extraída de ISO/IEC 15939 (2008)

.

quando todos os critérios de aceitação estiverem satisfeitos. Nesta atividade são refinados os critérios de aceitação que foram definidos no plano de projeto e incorporados no pedido de proposta e no contrato. As avaliações podem ser conduzidas no decorrer do contrato, por uma abordagem envolvendo múltiplas iterações e entregas de produtos, ou por meio de uma entrega única. Os S&SC entregues são analisados para identificar a conformidade aos critérios estabelecidos. As tarefas de avaliação são concebidas de modo a reduzir a interferência com as avaliações executadas pelo fornecedor e a duplicação de esforços de avaliação. Não havendo aprovação do S&SC, e dependendo das cláusulas contratuais, podem ser planejados e implementados ajustes para que o produto seja submetido a uma nova avaliação. Este ciclo ocorre enquanto o produto não é aprovado, ou até que seja definitivamente rejeitado. As tarefas previstas compreendem:

- **Definir critérios de aceitação**
- **Avaliar o produto entregue**
- **Manter conformidade com o contrato**
- **Aceitar o S&SC**

A Instrução Normativa Nº 04 trata do processo de contratação de serviços de TI pela Administração Pública Federal direta, autárquica e fundacional. Esta Instrução Normativa IN (Instrução Normativa) será explicada na próxima seção, onde os artigos mais relevantes no contexto da aquisição de *software* e da aceitação de *software* pelo cliente serão abordados.

5.2 Instrução Normativa Nº 04

Instruções normativas são atos expedidos por autoridades administrativas, normas complementares das leis, dos tratados e das convenções internacionais e dos decretos, e não podem transpor, inovar ou modificar o texto da norma que complementam. As instruções normativas visam regulamentar ou implementar o que está previsto nas leis que, no caso do Brasil, são apreciadas, elaboradas e aprovadas pelo Congresso Nacional, e sancionadas pelo Presidente da República.

A IN4 (2014) dispõe sobre o processo de contratação de Soluções de Tecnologia da Informação pelos órgãos integrantes do Sistema de Administração de Recursos de Tecnologia da Informação e Informática (SISP) do Poder Executivo Federal. Esta IN é a consolidação de um conjunto de boas práticas para Contratação de Solução de TI, que formam o Modelo de Contratações de Soluções de TI (MCTI).

A Instrução Normativa nº 04 IN4 (2014) está dividida em três capítulos:

- **Capítulo 1:** Diz respeito às disposições gerais.
- **Capítulo 2:** Diz respeito ao processo de contratação e é dividido em 3 seções.
  - **Seção 1:** Diz respeito ao Planejamento da Contratação e é dividido em 4 Subseções.
  - **Seção 2:** Diz respeito à Seleção do Fornecedor.
  - **Seção 3:** Diz respeito à Gestão do Contrato e é dividido em 4 Subseções.
- **Capítulo 3:** Apresenta as Disposições Finais.

Os artigos mais relevantes e que serão abordados neste trabalho são os artigos 20 e 34, ambos inseridos no capítulo 2 da referida normativa.

O caput do artigo 20 do IN4 (2014) trata sobre o modelo de gestão do contrato, definido a partir do Modelo de Execução do Contrato, que deverá contemplar as condições para gestão e fiscalização do contrato de fornecimento da Solução de Tecnologia da Informação.

O inciso II versa sobre os procedimentos de teste e inspeção, para fins de elaboração dos Termos de Recebimento Provisório e Definitivo. Ademais, na alínea "a", itens 1,2 e 5 tratam sobre a metodologia, formas de avaliação da qualidade e adequação da solução de Tecnologia da Informação às especificações funcionais e tecnológicas, observando:

- **Item 1:** Definição de mecanismos de inspeção e avaliação da Solução, a exemplo de inspeção por amostragem ou total do fornecimento de bens ou da prestação de serviços.

- **Item 2:** Adoção de ferramentas, computacionais ou não, para implantação e acompanhamento dos indicadores estabelecidos.

- **Item 5:** Garantia de inspeções e diligências, quando aplicáveis, e suas formas de exercício.

O inciso III versa sobre a fixação dos valores e procedimentos para retenção ou glosa no pagamento, sem prejuízo das sanções cabíveis, que só deverá ocorrer quando a contratada incindir nas alíneas "a"e "b"do mencionado inciso.

- **alíneas "a":** não atingir os valores mínimos aceitáveis fixados nos Critérios de Aceitação, não produzir os resultados ou deixar de executar as atividades contratadas; ou

- **alíneas "b":** deixar de utilizar materiais e recursos humanos exigidos para fornecimento da solução de tecnologia da informação, ou utilizá-los com qualidade ou quantidade inferior à demandada.

Por fim, no inciso IV trata sobre a definição clara e detalhada das sanções administrativas observadas, principalmente, nas alíneas "b"e "c".

- **alíneas "b":** proporcionalidade das sanções previstas ao grau do prejuízo causado pelo descumprimento das respectivas obrigações.

- **alíneas "c":** as situações em que advertências ou multas serão aplicadas, com seus percentuais correspondentes, que obedecerão a uma escala gradual para as sanções recorrentes.

Outrossim o artigo 34 da referida normativa trata sobre o monitoramento da execução deverá observar o disposto no Plano de Fiscalização da contratada e o disposto no modelo de gestão do contrato, observando os seguintes incisos:

- **II:** Avaliação da qualidade dos serviços realizados ou dos bens entregues e justificativas, a partir da aplicação das Listas de Verificação e de acordo com os Critérios de Aceitação definidos em contrato, a cargo dos Fiscais Técnico e Requisitante do Contrato.

- **III:** Identificação de não conformidade com os termos contratuais, a cargo dos Fiscais Técnico e Requisitante do Contrato.

- **VI:** Encaminhamento das demandas de correção à contratada, a cargo do Gestor do Contrato ou, por delegação de competência, do Fiscal Técnico do Contrato.

- **VII:** Encaminhamento de indicação de glosas e sanções por parte do Gestor do Contrato para a Área Administrativa.

6 *Data Warehouse*

Neste capítulo será apresentada a fundamentação teórica sobre *Data Warehouse* e como utiliza-lo para o monitoramento de métricas, *bugs* e violações. Também será apresentado o ambiente de *Data Warehouse* utilizado na solução proposta por Rêgo (2014), como ela foi desenvolvida e os incrementos no qual este trabalho buscou aumentar o valor da aferição da qualidade.

6.1 Definição

Na década de 80 as organizações perceberam a importância de não apenas usar dados para propósitos operacionais, mas também para derivar a inteligência por trás deles. Essa inteligência não só justificaria as decisões passadas, mas também ajudaria na tomada de decisões para o futuro. O termo *Business Intelligence* tornou-se cada vez mais popular, e foi durante o final dos anos 1980 que pesquisadores da IBM, Barry Devlin e Paul Murphy, desenvolveram o conceito de *Business data warehouse* . A partir que aplicações de *business intelligence* foram surgindo, foi rapidamente verificado que os dados de bancos transacionais tinham que primeiramente ser transformados e armazenados em outros bancos de dados com um esquema específico para poderem derivar de sua inteligência. Esta base de dados poderia ser usada para arquivamento, e seria maior em tamanho do que as bases de dados transacionais, mas seu *design* seria ideal para executar relatórios que permitem as grandes organizações planejarem e tomarem decisões de forma proativa. Este banco de dados, normalmente armazenando as atividades realizadas no passado e no presente das organizações, foi chamado de *Data Warehouse* (SHARMA, 2011).

Para Inmon (2002), *Data Warehouse* é uma coleção de dados que tem como característica ser orientada a assunto, integrada, não volátil e temporal. Por orientação a assunto, podemos entender como um foco em algum aspecto específico da organização. O fato do ambiente ser integrado remete ao fato dele ser alimentado com dados que têm como origem múltiplas fontes, integrando esses dados de maneira a construir uma única orientação. Como um conjunto não volátil e temporal de dados, é entendido que a informação carregada remete a um determinado momento da aplicação, possibilitando assim acesso a diferentes intervalos de tempo, não havendo como modificá-los atualizando em tempo real.

Segundo Rocha (2000), *Data Warehousing* é a infra-estrutura tecnológica de *hardware* e *software* para a atividade de análise gerencial. Agora que sabemos o que é o *data wharehouse* e *Data Warehousing*, serão mostrado os componentes que compõem um ambiente completo de *data warehousing*. É importante entender como os componentes

funcionam individualmente antes de começarem a ser combinados para se criar um *data warehouse*. Cada componente do armazém tem uma função específica. É preciso aprender a importância estratégica de cada componente e como manuseá-los efetivamente para fazer uso do *data warehousing*. Uma das maiores ameaças ao sucesso no *data warehousing* é confundir os papéis e as funções dos componentes (KIMBALL; ROSS, 2002). A Figura 5 descreve uma arquitetura geral de um ambiente de *Data Warehousing*, os componentes do ambiente serão esclarecidos no decorrer deste capítulo.

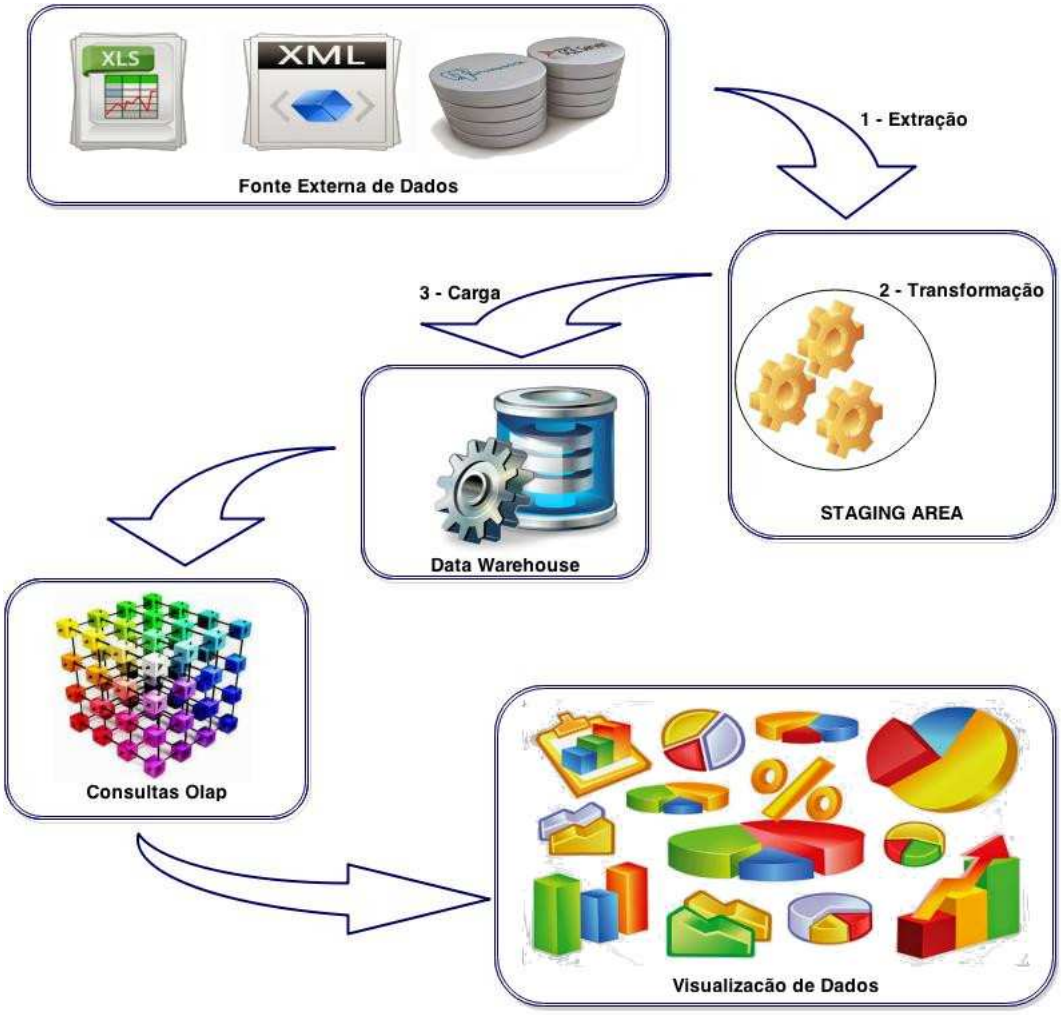

Figura 5 – Arquitetura de um ambiente de Data Warehousing

6.2 *Extraction-Transformation-Load*

A *Staging Area* é uma área de armazenamento onde acontece o processo de ETL. Na Figura 5, as etapas 1- Extração, 2- Transformação e 3- Carga formam o processo de *Extraction-Transformation-Load* (ETL). Cada uma das etapas recebe a seguinte descrição:

- **Extração:** Primeira etapa do processo de ETL, consiste na leitura e entendimento da fonte dos dados, copiando os que são necessários para futuros trabalhos (KIM-

BALL; ROSS, 2002).

- **Transformação:** Após a etapa de extração ter sido feita, os dados podem receber diversos tipos de transformações, que incluem correções de conflitos, conversão de formatos, remoção de campos que não são úteis, combinação entre dados de diversas fontes, entre outros (KIMBALL; ROSS, 2002).

- **Carga:** Após ter sido realizado o processo de transformação, os dados já estão prontos para serem carregados no *Data Warehouse*, tornando possível que todos os dados visualizados após esse processo reflitam a informação que passou pelos processos de extração e transformação (SHARMA, 2011).

6.3 Modelagem Dimensional

Modelagem dimensional é um novo nome para uma velha técnica para deixar os bancos de dados simples e compreensíveis. No início dos anos 70 as organizações de TI, consultores, usuários finais e fornecedores tiveram que migrar para uma estrutura dimensional simples que combinasse com a necessidade humana pela simplicidade (KIMBALL; ROSS, 2002).

Segundo Kimball e Ross (2002), a modelagem dimensional tem sido amplamente aceita como a técnica dominante para a apresentação do *data warehouse*. Os profissionais e especialistas de *data warehouse* reconhecem que a apresentação do *data warehouse* deve ser fundamentada na simplicidade. A simplicidade é a chave fundamental que permite que os usuários entendam facilmente as bases de dados e naveguem de forma eficiente no bancos de dados do *software*.

Para facilitar na difusão do conceito de modelagem dimensional Kimball e Ross (2002) utiliza como exemplo um diretor geral que descreve seus negócios como "Nós vendemos produtos em várias áreas de negócio e medimos nosso desempenho ao longo do tempo". Assim, designers dimensionais colocariam em ênfase o produto, as áreas de negócio e o tempo, pensando intuitivamente neste negócio como um cubo de dados, onde as bordas estariam marcadas como produto, negócio e tempo. Pontos dentro do cubo são onde as medições que combinam produtos, áreas de negócio e tempo são salvas. A capacidade de visualizar algo tão abstrato como um conjunto de dados de uma forma concreta e tangível é o segredo da compreensão.

Os conceitos básicos da modelagem dimensional são: fatos, dimensões e medidas. Um fato é uma coleção de itens de dados relacionados, que consiste em medidas e dados de contexto. Ele representa tipicamente itens de negócios ou transações de negócio. A dimensão é uma coleção de dados que descrevem uma dimensão negócio. Dimensões determinam o contextual a fundo para os fatos; eles são os parâmetros nos quais queremos

realizar a *On-Line Analytic Processing (OLAP)*. A medida é um atributo numérico de um fato, que representa o desempenho ou comportamento do negócio em relação às dimensões (GUTIéRREZ; MAROTTA, 2000).

Considerando o contexto relacional, existem dois modelos básicos que são utilizados na modelagem dimensional: modelo de estrela e modelo de floco de neve. O modelo estrela é a estrutura básica de um modelo dimensional. Ele tem uma grande tabela central (tabelas fato) e um conjunto de pequenas tabelas (tabelas dimensão) dispostos em um padrão radial ao redor da tabela central, como é mostrado na Figura 6. O modelo de floco de neve é o resultado da decomposição de uma ou mais das dimensões (GUTIéRREZ; MAROTTA, 2000).

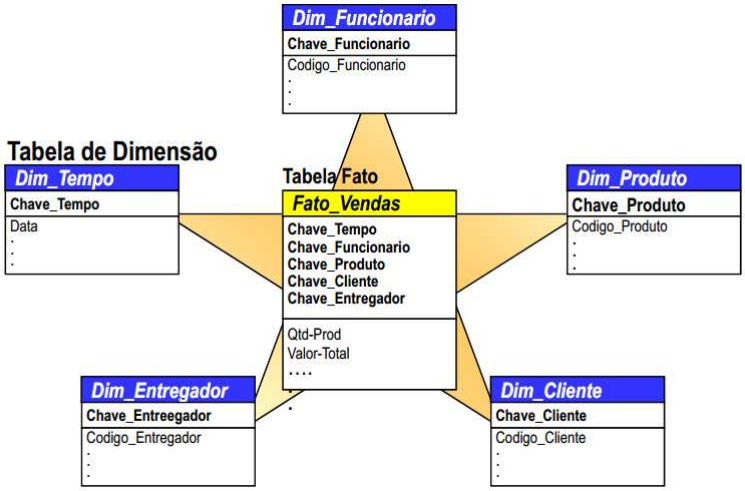

Figura 6 – Esquema estrela extraído de Times (2012)

6.3.1 OLAP *(On-Line Analytic Processing)*

A atividade que consiste em buscar e apresentar os dados de um *Data Warehouse*, sendo essa busca quase sempre baseada em um cubo multidimensional de dados, é chamada de *On-Line Analytic Processing* (OLAP) (KIMBALL; ROSS, 2002). Segundo Sharma (2011), *On-Line Analytic Processing(OLAP)* refere-se a cargas onde grandes quantidades de históricos de dados são processados para gerar relatórios e executar análise de dados. Normalmente, bancos de dados que utilizam OLAP são alimentados a partir de bancos de dados *On-Line Transaction Processing* (OLTP) que são alterados para gerir cargas OLAP. Um banco de dados OLAP armazena um grande volume com os mesmos dados transacionais que um banco de dados OLTP, mas estes dados são transformados pelo processo de ETL para permitir um melhor desempenho para geração de relatórios e para facilitar as análises. Geralmente sistemas OLTP são ajustados para inserções, atualizações e exclusões bem rápidas, enquanto os sistemas OLAP estão ajustados para consultas rápidas.

A tabela 13 evidencia as diferenças entre aplicações OLTP e OLAP extraídas do trabalho de (NERI, 2002):

OLPT	OLAP
Controle operacional	Tomada de decisão
Atualização de dados	Análise de dados
Pequena complexidade das operações	Grande complexidade das operações
Não armazena dados históricos	Armazena dados históricos
Voltada ao pessoal operacional	Voltada aos gestores do negócio

Tabela 13 – Diferenças entre OLTP e OLAP extraídas de Neri (2002)

No modelo multidimensional, os dados são organizados em múltiplas dimensões, e cada dimensão contém vários níveis de abstração definidos pelas hierarquias. Esta organização fornece aos usuários a flexibilidade para visualizar os dados a partir de diferentes perspectivas. Existe uma série de operações de cubo de dados OLAP para materializar esses diferentes pontos de vista, permitindo consultas interativas e análises de dados. Assim, OLAP fornece um ambiente amigável para análise de dados interativos.

Entre as operações OLAP estão:

- ***Drill Down:*** Busca aumentar o nível de detalhamento, partindo de um certo nível de dados para um nível mais detalhado (SHARMA, 2011).
- ***Drill Up:*** Ao contrário da operação *Drill Down*, a *Roll Up* parte de um nível mais detalhado para um nível menos detalhado (SHARMA, 2011).

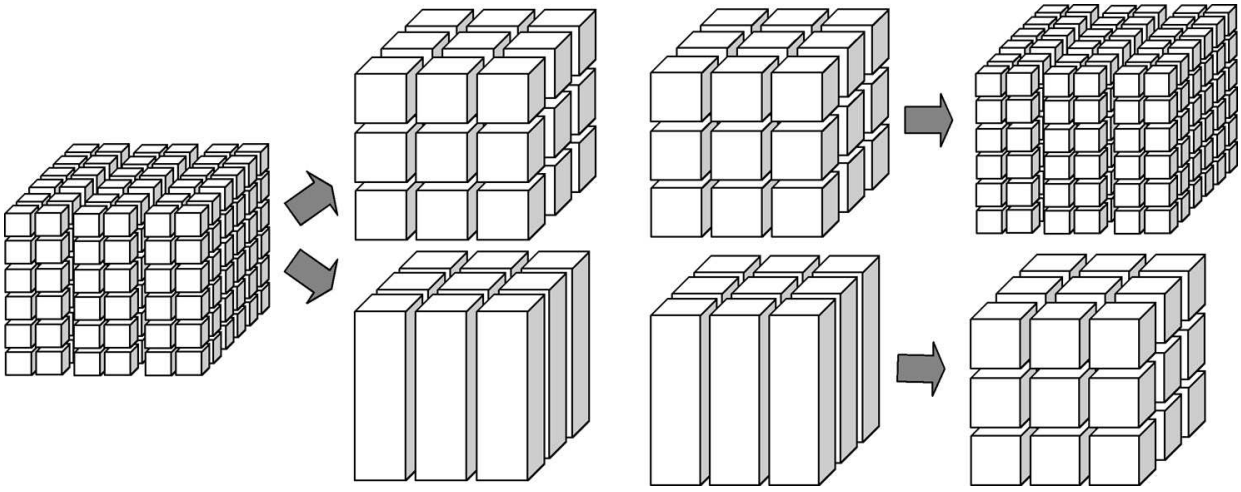

Figura 7 – Exemplo de operações *Drill Down*(direita) e *Drill up*(esquerda) extraídos de Golfarelli (2009)

- ***Slice and Dice:*** Técnica com filosofia parecida à cláusula *where* usada em *SQL*. Permite que sejam criadas restrições na análise dos dados (TIMES, 2012). O *Slice*

faz restrição de um valor ao longo de uma dimensão, já o *Dice* faz restrições de valores em várias dimensões. Semelhante ao *Slice*, só que mais complexo.

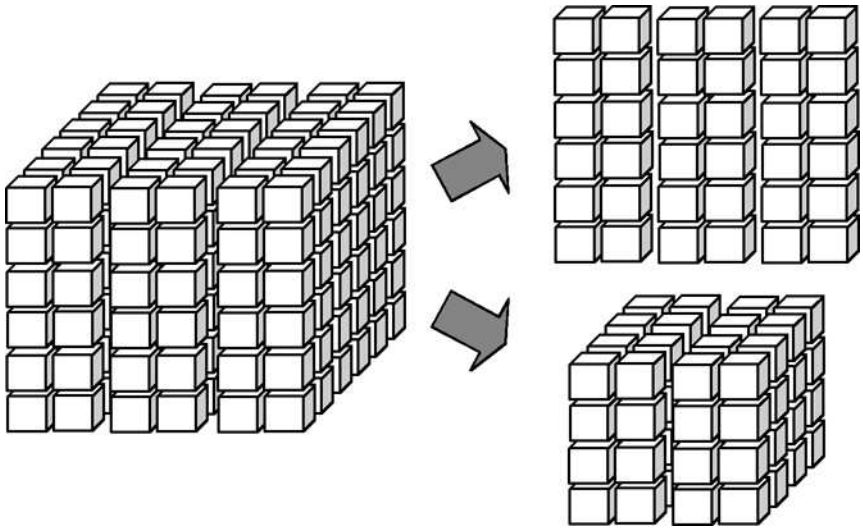

Figura 8 – Exemplo de operações *Slice*(acima) e *Dice*(embaixo) extraídos de Golfarelli (2009)

- ***Drill Across:*** Permite que diferentes cubos sejam concatenados (NERI, 2002). Uma operação do tipo *Drill Across* irá simplesmente unir diferentes tabelas fato através de dimensões correspondentes (KIMBALL, 1998).

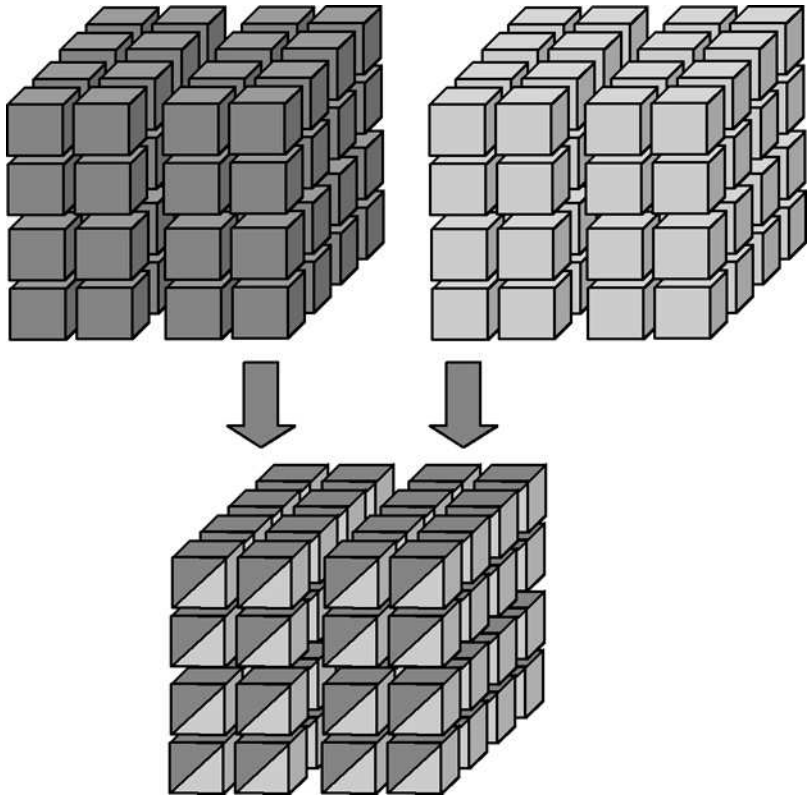

Figura 9 – Exemplo da operação *Drill Across* extraído de Golfarelli (2009)

- ***Pivoting:*** Metaforicamente, significa rotacionar o cubo. Essa técnica altera a ordenação das tabelas dimensionais (NERI, 2002).

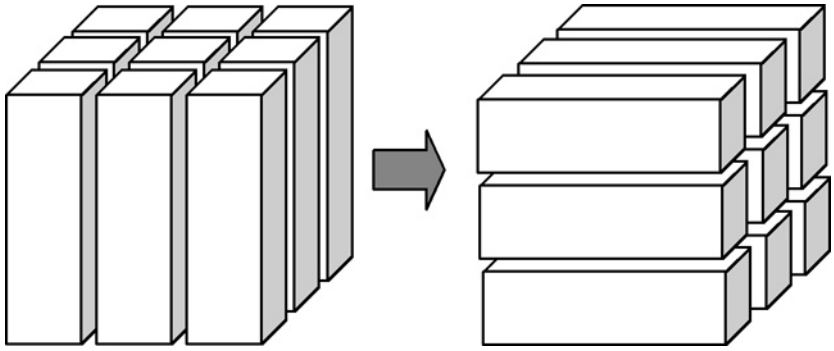

Figura 10 – Exemplo da operação *Pivoting* extraído de Golfarelli (2009)

6.4 Ambiente de *Data Warehousing* para Métricas, *bugs* e violações de Código-Fonte

Para a implementação do ambiente de Data Warehousing para métricas de código-fonte, Rêgo (2014) definiu a arquitetura tal como mostra a Figura 5.

A ferramenta de análise estática de código-fonte escolhida por Rêgo (2014) foi a Analizo. A Analizo possibilita emitir saídas das métricas em CSV que detalham nome da classe e as respectivas métricas, o que permitiu a seu trabalho incorporar a análise das métricas ANPM, AMLOC, CBO, NPA. As ferramentas FindBugs e PMD, descritas no capítulo 3, foram adicionadas ao ambiente de *Data Warehousing* como pode ser visto na Figura 17.

Rêgo (2014) seguiu a metodologia de (KIMBALL; ROSS, 2002), apresentada na Figura 11, para o seu projeto de *data warehouse*. Seguindo esta metodologia, Rêgo (2014) definiu os seguintes requisitos de negócio que a sua solução deveria suportar:

- **Requisito 1:** Visualizar o intervalo qualitativo obtido para cada métrica de código-fonte em uma determinada *release* do projeto para a configuração O*pen JDK8 Metrics*.
- **Requisito 2:** Comparar o intervalo qualitativo obtido para cada métrica de código-fonte ao longo de todas as *releases* de um projeto para a configuração *Open JDK8 Metrics*.
- **Requisito 3:** Visualizar o valor percentil obtido para cada métrica de código-fonte em uma determinada *release* do projeto para a configuração *Open JDK8 Metrics*.
- **Requisito 4:** Comparar o valor percentil de cada métrica de código-fonte ao longo de todas as *releases* para a configuração *Open JDK8 Metrics*.
- **Requisito 5:** Visualizar o intervalo qualitativo obtido para cada métrica de código-fonte em uma determinada *release* do projeto para a configuração *Tomcat Metrics*.
- **Requisito 6:** Comparar o intervalo qualitativo obtido para cada métrica de código-fonte ao longo de todas as *releases* de um projeto para a configuração *Tomcat Metrics*.
- **Requisito 7:** Visualizar a medida obtida para cada métrica de código-fonte em uma determinada *release* do projeto para a configuração *Tomcat Metrics*.
- **Requisito 8:** Comparar o valor percentil obtido para cada métrica de código-fonte ao longo de todas as *releases* para a configuração *Tomcat Metrics*.
- **Requisito 9:** Visualizar a quantidade de cenários de limpeza identificados por tipo de cenários de limpeza de código-fonte em cada classe ao longo de cada *release* de um projeto.
- **Requisito 10:** Comparar a quantidade de cenários de limpeza por tipo de cenários de limpeza de código-fonte em uma *release* de um projeto.

- **Requisito 11:** Visualizar o total de cenários de limpeza em uma determinada *release* de um projeto.
- **Requisito 12:** Visualizar cada uma das classes com um determinado cenário de limpeza de código-fonte ao longo das *releases* do projeto.
- **Requisito 13:** Visualizar as 10 classes de um projeto com menor número de cenários de limpeza identificados.
- **Requisito 14:** Visualizar as 10 classes de um projeto com maior número de cenários de limpeza identificados.
- **Requisito 15:** Acompanhar a Taxa de Aproveitamento de Oportunidades de Melhoria de Código-Fonte que é a divisão do total de cenários de limpeza identificados em uma *release* e o o número total de classes da mesma *release* de um projeto

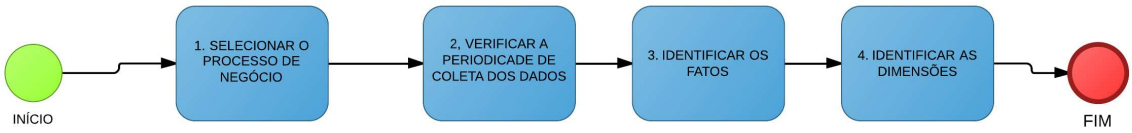

Figura 11 – Metodologia de Projeto de *Data Warehouse* proposta por Kimball e Ross (2002) extraída de Rêgo (2014)

Utilizando a mesma metodologia que o Rêgo (2014), foi definido novos requisitos de negócio que a solução deveria suportar:

- Requisito 16: Visualizar o total de *bugs* em uma determinada *release* de um projeto.
- Requisito 17: Visualizar o total de violações em uma determinada *release* de um projeto.
- Requisito 18: Visualizar a quantidade de *bugs* por severidade em uma determinada *release* de um projeto.
-
- Requisito 19: Visualizar a quantidade de *bugs* por tipo em uma determinada *release* de um projeto.
- Requisito 20: Visualizar a quantidade de *bugs* por classe em uma determinada *release* de um projeto.
- Requisito 21: Visualizar a quantidade de violações por severidade em uma determinada *release* de um projeto.
- Requisito 22: Visualizar a quantidade de violações por tipo em uma determinada

release de um projeto.

- Requisito 23: Visualizar a quantidade de violações por classe em uma determinada *release* de um projeto.

Ainda seguindo a metodologia de (KIMBALL; ROSS, 2002), Rêgo (2014) identificou fatos e dimensões no contexto de monitoramento de métricas, como mostra a Tabela 14.

Fato	Dimensões
Valor Percentil	• Projeto • Métrica • Configuração • Qualidade • *Release* • Tempo
Quantidade de Cenários de Limpeza	• Projeto • Cenário de Limpeza • Classe • *Release*
Taxa de Aproveitamento de Oportunidade de Melhoria de Código-Fonte	• Projeto • *Release*

Tabela 14 – Fatos e dimensões identificadas por Rêgo (2014)

Dando prosseguimento a metodologia de (KIMBALL; ROSS, 2002), foram identificadas mais fatos e dimensões em complemento à solução de Rêgo (2014), como mostra a Tabela 15.

Fato	Dimensões
FindBugs	• Projeto • *Release* • Tempo • Classe do *Bug* • Tipo do *Bug* • Prioridade do *Bug*
PMD	• Projeto • Tempo • *Release* • Classe da Violação • Regras • Severidade

Tabela 15 – Fatos e dimensões identificadas neste trabalho.

Após a identificação dos fatos e das dimensões, Rêgo (2014) construiu o projeto físico do *Data Warehouse* que pode ser visto na Figura 12. A Tabela 16 facilita a interpretação do projeto físico.

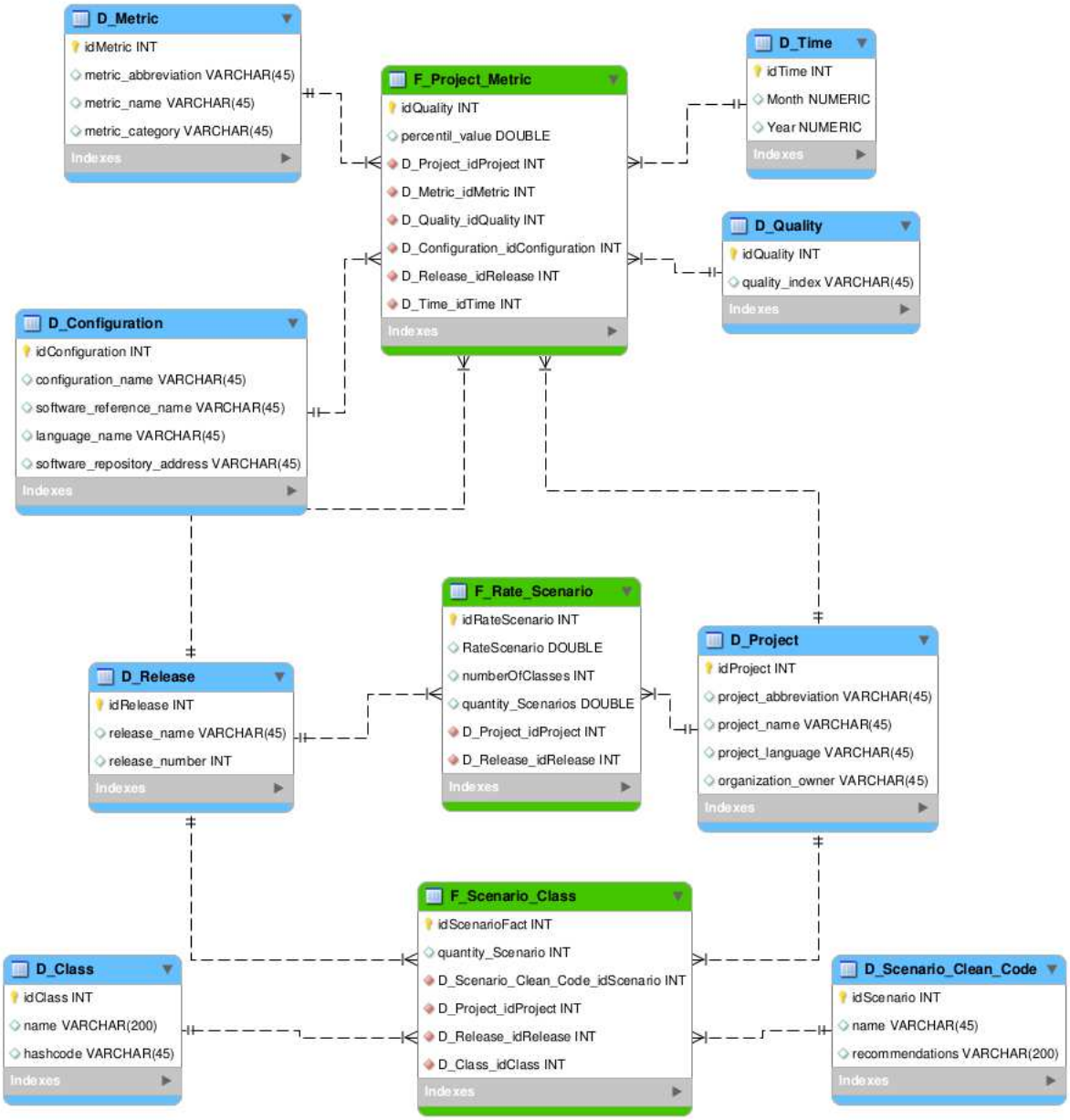

Figura 12 – Projeto físico do *Data Warehouse* extraído de Rêgo (2014)

Tabela Fato	Tabelas Dimensões
F_Project_Metric	• D_Project • D_Metric • D_Configuration • D_Quality • D_Release • D_Time
F_Scenario_Class	• D_Project • D_Scenario_Clean_Code • D_Class • D_Release
F_Rate_Scenario	• D_Project • D_Release

Tabela 16 – Tabelas fatos e tabelas dimensões elaboradas por Rêgo (2014)

Visando facilitar o processo de ETL principalmente na etapa da transformação, Rêgo (2014) criou uma área de metadados mostrada na Figura 13 com intuito de tratar os dados que representam os próprios dados dos processos de negócio. A Tabela 17 descreve as tabelas contidas na área de metadados do *Data Warehouse*.

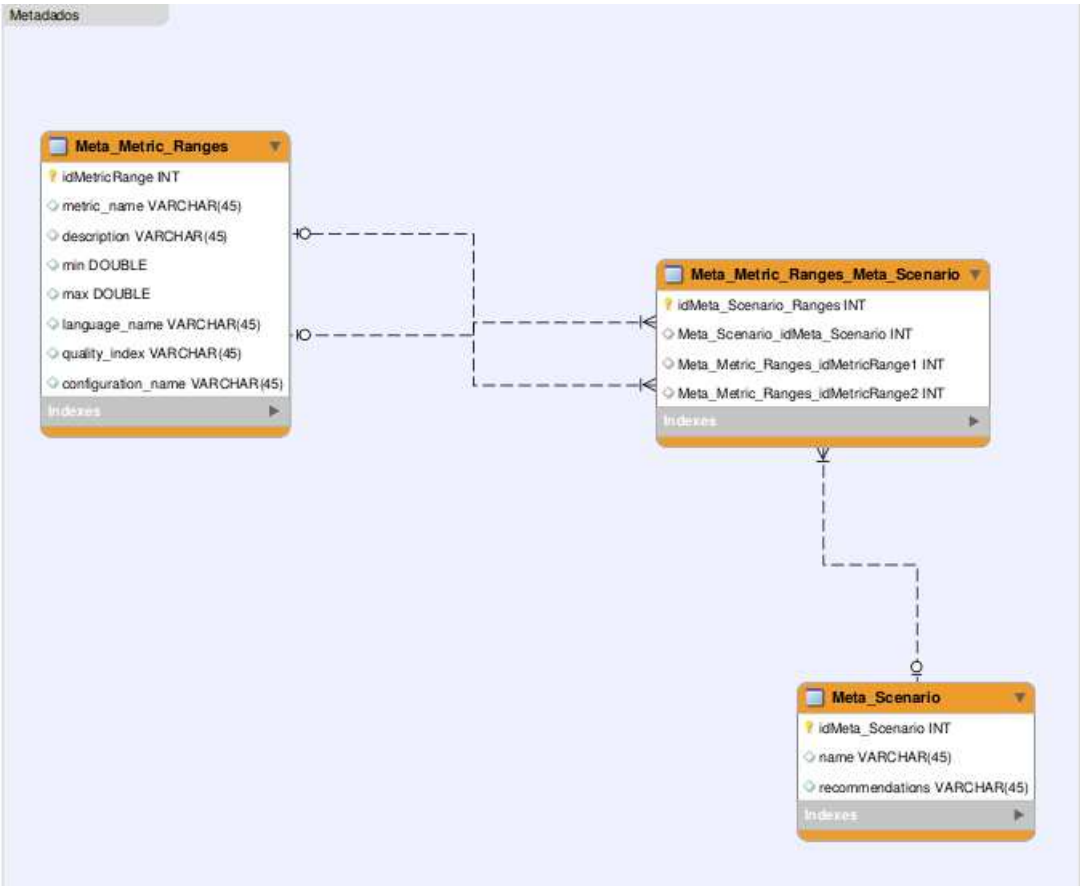

Figura 13 – Projeto físico do *Data Warehouse* extraído de Rêgo (2014)

Tabelas	Descrição
Meta_Metric_Ranges	Contém cada configuração de intervalo qualitativo para cada métrica de código fonte.
Meta_Scenario	Contém os cenários de limpeza de código e suas recomendações.
Meta_Metric_Ranges_Meta_Scenario	Como a cardinalidade entre as tabelas *Meta_Scenario* e *Meta_Metric_Ranges* seria de n para n, essa tabela foi criada contendo os registros únicos dessas duas tabelas.

Tabela 17 – Descrição das Tabelas do Metadados do *Data Warehouse*

Após a identificação dos fatos e das dimensões deste trabalho, o projeto físico do *Data Warehouse* construído por Rêgo (2014) foi alterado. Foram adicionadas as Tabelas *F_Project_Bug, D_Class_Bug, D_Type, D_Priority, F_Project_Violation, D_Rules, D_Severite e D_Class_Violation* como pode ser visto nas Figuras 14 e 15. A Figura 16

mostra como ficou o projeto físico finalizado e as Tabelas 16, 18 e 17 facilitam a sua interpretação.

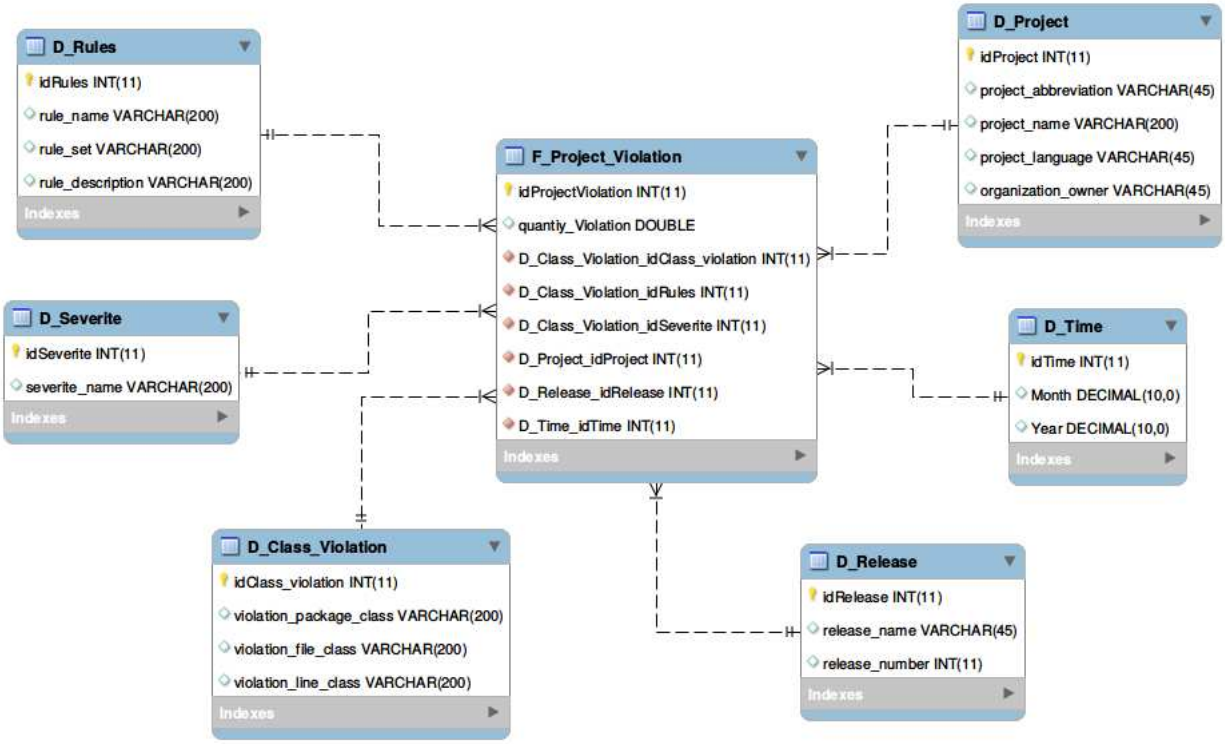

Figura 14 – Parte do projeto físico contendo as tabelas relacionados ao PMD

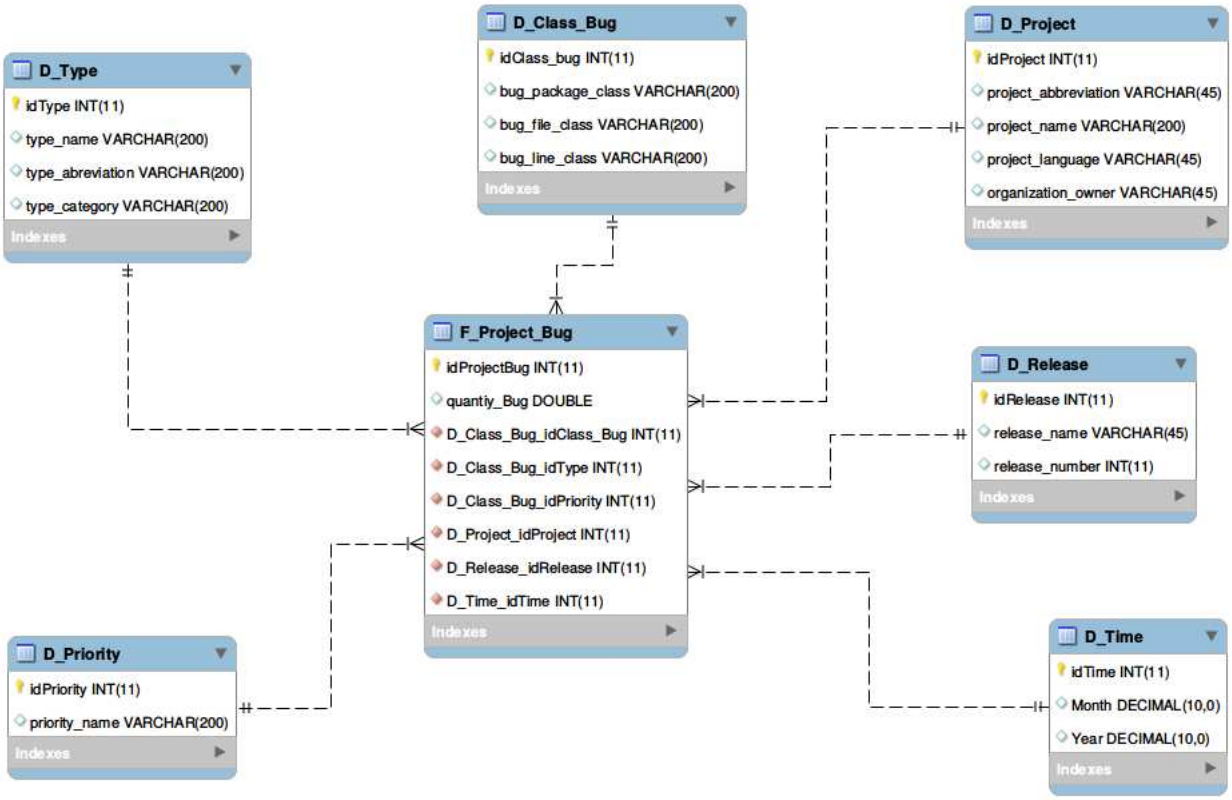

Figura 15 – Parte do projeto físico contendo as tabelas relacionados ao *FindBugs*

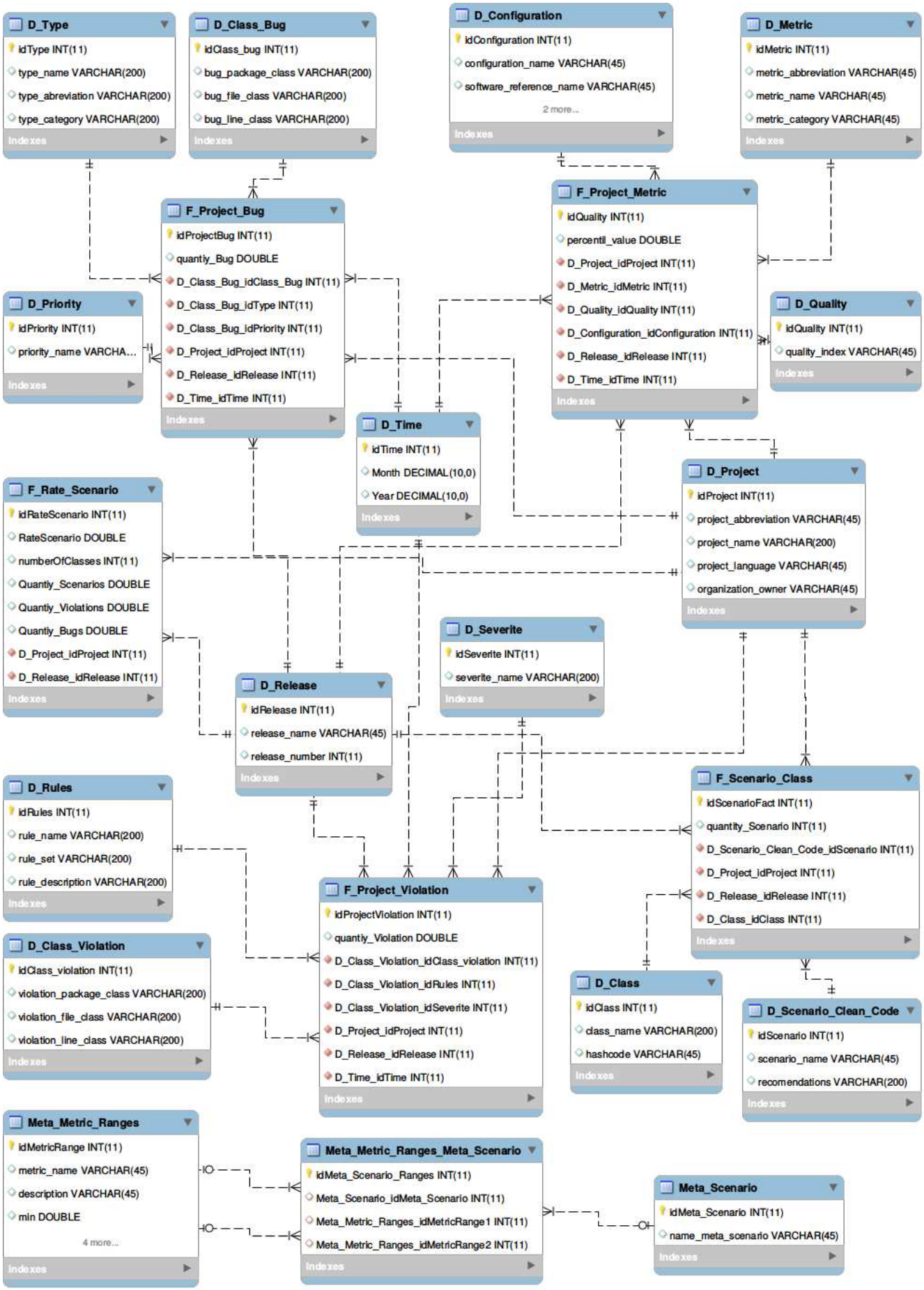

Figura 16 – Projeto físico do *Data Warehouse* finalizado.

Tabela Fato	Tabelas Dimensões
F_Project_Bug	• D_Project • D_Release • D_Time • D_Class_Bug • D_Type • D_Priority
F_Project_Violation	• D_Project • D_Release • D_Time • D_Class_Violation • D_Rules • D_Severite

Tabela 18 – Tabelas fatos e tabelas dimensões adicionadas à solução de Rêgo (2014)

6.4.1 Ferramentas de *Data Warehousing*

Entre as alternativas de código aberto que suportam um modelo dimensional, Rêgo (2014) utilizou o *Pentaho Business Analytics Community Edition*. Esta alternativa livre apresenta soluções que cobrem as áreas de ETL, *reporting*, *OLAP* e mineração de dados. A Figura **??**, apresenta o modo como cada uma das ferramentas está disposta na arquitetura do ambiente de *Data Warehousing* para Métricas de Código-Fonte.

Segue abaixo as ferramentas utilizadas em cada etapa do *Data Warehouse*:

- **ETL:** Pentaho Data Integration Community Edition ou Kettle.
- **Implementação das Consultas OLAP e Visualização de Dados:** *Pentaho BI Platform 8*, provê uma arquitetura e a infraestrutura para soluções de *Business Inteligence*, *Data Mining* e a camada de visualização de dados do *Data Warehouse*. Junto ao *Pentaho BI Platform 8* foi utilizado o *plugin Saiku Analytics* que oferece serviços de apoio a operações OLAP e à visualização de dados. Na arquitetura do *Saiku Analytics*, está incorporado outro software livre que realiza consulta, chamado de *Mondrian OLAP*.
- **Análise estática de código-fonte:***Analizo*, possibilita emitir saídas das métricas em CSV que detalham o nome da classe e as suas respectivas métrica.

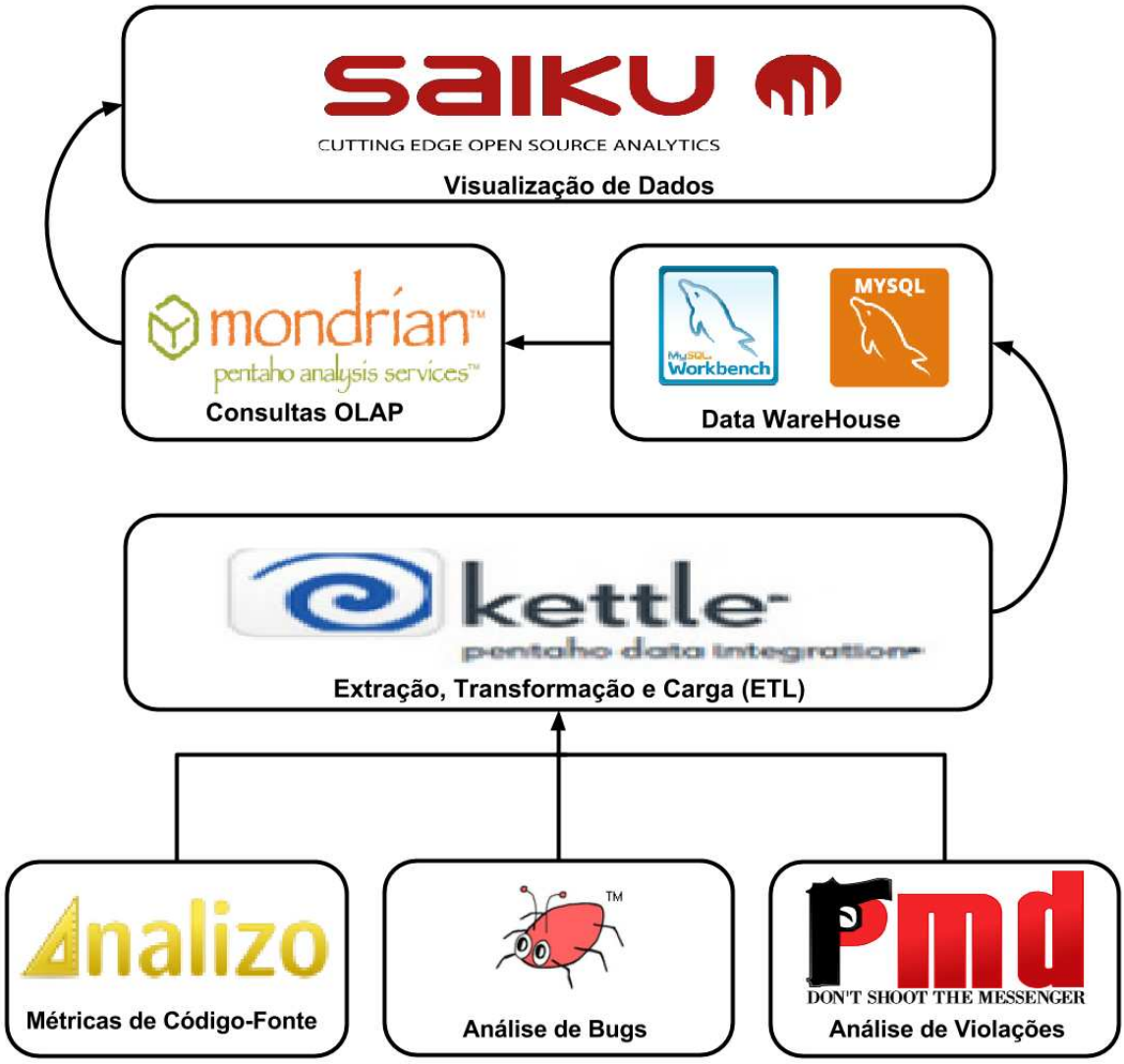

Figura 17 – Arquitetura Ambiente de *Data Warehousing* para Métricas de Código-Fonte extraído de Rêgo (2014) e adaptado.

6.5 Considerações finais do capítulo

Nesse capítulo foi apresentada a solução proposta no trabalho de Rêgo (2014), bem como a base teórica para sua compreensão. No próximo capítulo será apresentado o projeto de estudo de caso que visa a análise da eficácia e eficiência da solução de DW proposta nesse capítulo.

7 Projeto de estudo de Caso

Neste capítulo é apresentado o projeto do estudo de caso. Isso consiste na elaboração de um protocolo para o estudo de caso, na identificação do problema e na definição das questões e objetivos de pesquisa. O método para coleta dos dados e como eles serão analisados também serão apresentados neste capítulo.

7.1 Definição

Segundo Yin (2001), o estudo de caso é um conjunto de procedimentos pré-especificados para se realizar um estudo empírico que investiga um fenômeno contemporâneo dentro de seu contexto da vida real, especialmente quando os limites entre o fenômeno e o contexto não estão claramente definidos. Uma grande vantagem do estudo de caso é a sua capacidade de lidar com uma ampla variedade de evidências, documentos, artefatos, entrevistas e observações. Além disso, em algumas situações, como na observação participante, pode ocorrer manipulação informal.

Buscando maior entendimento a respeito do estudo de caso proposto, foram criadas algumas perguntas que são fundamentais para o seu entendimento:

- Qual o escopo do estudo de caso?
- Qual o problema a ser tratado?
- Qual a questão de pesquisa relacionada a esse problema?
- Quais são os objetivos a serem alcançados nessa pesquisa?
- Como foi a seleção do estudo de caso?
- Qual é a fonte dos dados coletados nessa pesquisa e qual o método de coleta?

Com o objetivo da elucidação do escopo do estudo de caso proposto neste trabalho bem como foi ilustrado na Figura 18, foram apresentados, nos capítulos anteriores: um estudo teórico relacionado à métricas de software, contratação de serviços de TI por parte da Administração Pública Federal brasileira e de Data *Warehouse*. Adicionalmente foi apresentada uma solução para o monitoramento de Métricas de Código-Fonte com suporte de um ambiente de Data *Warehousing*.

Para as demais perguntas a serem respondidas, foi estruturado neste trabalho um protocolo de estudo de caso baseado em Brereton, Kitchenham e Budgen (2008) que é dividido da seguinte maneira:

- **Background - Seção 7.2**: Identificar outros estudos acerca do tópico, definir a questão de pesquisa principal e suas proposições derivadas que serão abordadas por este estudo.

- **Design - Seção 7.3**: Identificar se o projeto de pesquisa é um caso único ou múltiplo bem como seu propósito geral. O estudo de caso único envolve a estratégia de pesquisa aplicada à compreensão de várias dimensões do fenômeno com foco em um caso singular enquanto o estudo de caso múltiplo envolve o estudo das mesmas dimensões do fenômeno em mais de um caso simultaneamente.

- **Seleção - Seção 7.4**: Apresentar critérios para a seleção do caso e descrição do objeto de estudo a ser analisado.

- **Fonte e Método de Coleta de Dados - Seção 7.5**: Identificar os dados que serão coletados, definindo um plano para a coleta e como a informação será armazenada.

- **Processo de Análise dos Dados - Seção 7.6**: Identificar os critérios para interpretação dos resultados do estudo de caso, relacionar os dados com a questão de pesquisa e elaborar a explicação do encontrado.

- **Ameaças a validade do estudo de caso - Seção 7.7**: Elicitar tipos de validades aplicáveis a um estudo de caso, baseando-se no trabalho desenvolvido por Yin (2001), sendo elas: constructo, interna, externa e confiabilidade.

- **Cronograma - Seção 7.8**: Cronograma com as estimativas de tempo para as atividades propostas.

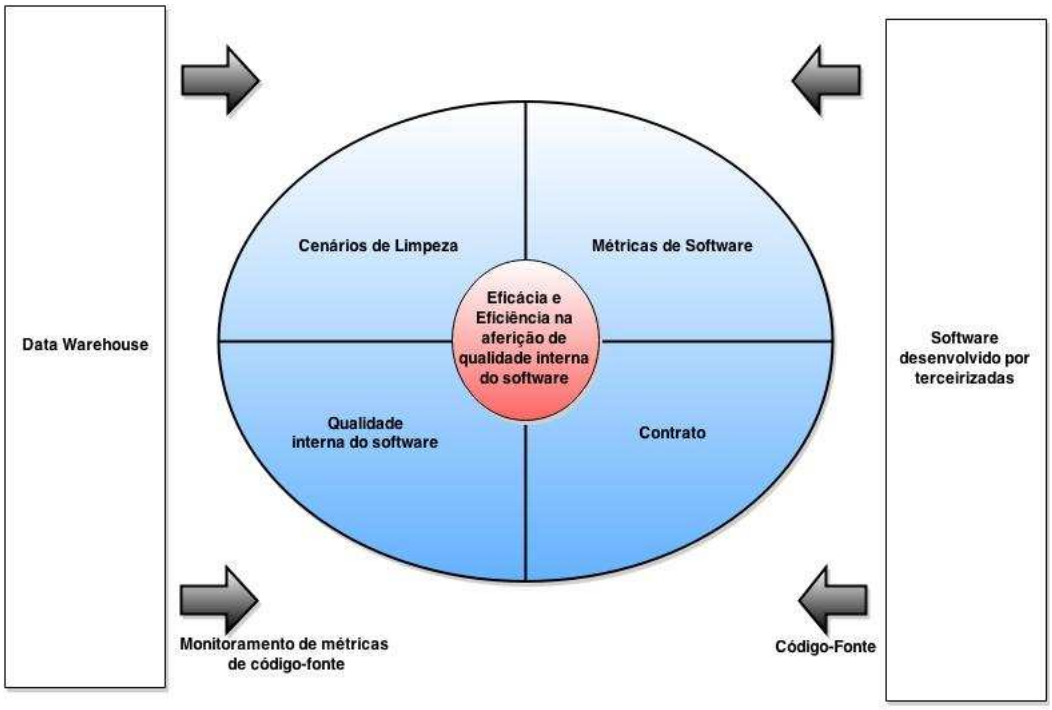

Figura 18 – Escopo do Estudo de Caso

7.2 Background

Esta seção contém referências sobre os trabalhos que antecederam esse estudo de caso dentro de um contexto similar ao que foi apresentado, assim como a própria questão geral de pesquisa a ser respondida com todos os elementos necessários para respondê-la.

7.2.1 Trabalhos Antecedentes e Relacionados

O principal trabalho que antecede essa ideia foi desenvolvido por Rêgo (2014), no qual a solução para monitoramento de métricas de código fonte utilizando *Data Warehouse* foi desenvolvida.

Anteriormente ao trabalho realizado por Rêgo (2014), Marinescu (2005) mostrou em seu trabalho que a utilização de métricas isoladas dificulta a interpretação de anomalias do código, reduzindo a aplicabilidade da medição feita. Além disso, o autor ainda afirma que a métrica por si só não contém informação o suficiente para motivar uma transformação no código que que melhore sua qualidade. Buscando trabalhar nesse contexto, Marinescu (2005) apresentou o conceito de interpretação das métricas em um nível de abstração maior que o adquirido ao observar apenas o valor da métrica.

Outro trabalho antecedente a ser destacado é a tese desenvolvida por Meirelles (2013), em que se buscou responder como métricas de código-fonte podem influir na atratividade de projetos de software livre e quais métricas devem ser controladas ao longo do tempo. Além disso, Meirelles (2013) também inseriu como questão de pesquisa se

as métricas de código-fonte melhoram com o amadurecimento dos projetos. Durante a execução de seu trabalho, foi observado que, a partir dos resultados obtidos para cada métrica de código-fonte em uma análise realizada no código-fonte de trinta e oito projetos de software livre, para uma determinada linguagem de programação, é possível definir um intervalo qualitativo a partir de um determinado percentil ao observar o conjunto de valores que são frequentemente obtidos.

O trabalho de Machini et al. (2010) fortemente relaciona-se com este trabalho por ter como objetivo fazer com que as métricas de código-fonte sejam mais facilmente incorporadas no cotidiano dos programadores, apresentando uma maneira de interpretar os valores das métricas através de cenários problemáticos que ocorrem frequentemente durante o desenvolvimento.

Já o trabalho de Ruiz et al. (2005) apresenta um ambiente de *data warehousing* para apoiar a implementação de um programa de medição em uma organização certificada como CMM Nível 2. Além disso, foi concebido um experimento que valida sua arquitetura, onde foram usados dados organizacionais reais. Ruiz et al. (2005) aborda três aspectos essenciais:

- A captura de dados, considerando os vários tipos de heterogeneidade.
- A integração, transformação e representação de dados quantitativos do projeto de acordo com a visão organizacional unificada e centralizada.
- A funcionalidade de análise que permite o monitoramento do processo.

Novello (2006) também utilizou *data warehouse* para monitoramento de métricas no processo de desenvolvimento de software, porém essas métricas não eram de código fonte.

Outros trabalhos relacionados ao conceito de métricas de software utilizados nessa revisão são Fenton e Pfleeger (1998), Kan (2002), McCabe (1976). Outros trabalhos que possuem como tema *data warehousing* foram utilizados durante a revisão bibliográfica, entre eles estão Inmon (2002), Kimball e Ross (2002), Sharma (2011).

7.2.2 Questão de Pesquisa

A questão de pesquisa a seguir foi proveniente do problema da falta de capacidade da administração pública em aferir, de forma sistemática e automatizada, a qualidade interna dos produtos de software adquiridos e desenvolvidos por organizações contratadas, que foi identificado na seção 1.3 do capítulo 1.

O uso de um ambiente de DW para aferição da qualidade interna do software apoiado pela análise de cenários de limpeza, análise de bugs e análise de violações pode assistir ao processo de aferição da qualidade de software?

Para estruturar a questão de pesquisa foi utilizada a técnica *goal-question-metrics* conforme ilustrado na figura 19. Segundo (BASILI; CALDIERA; ROMBACH, 1996), o resultado da aplicação do GQM é a especificação de um sistema de medida destinada a um conjunto especifico de questões e um conjunto de regras para a interpretação dos dados da medição. O GQM é uma estrutura hierárquica que começa com um objetivo que especifica o propósito da medição, objeto a ser medido, questão a ser medida e o ponto de vista de quem a observa. O objetivo é refinado em várias questões, onde para cada questão é definida uma métrica(objetiva ou subjetiva)(BASILI; CALDIERA; ROMBACH, 1996). A estrutura do GQM está a seguir.

Objetivo 01: Avaliar a eficácia e eficiência do uso de *Data Warehouse* para monitoramento de métricas de código fonte.

QE01: Quantas tomadas de decisão foram realizadas pela equipe baseando-se no uso da solução desenvolvida em um período de tempo?

Fonte: Registro de observação em campo.

Métrica: Número de decisões tomadas/tempo.

QE02: Quantas tomadas de decisão ao todo foram realizadas pela equipe baseando-se no uso da solução desenvolvida?

Fonte: Registro de observação em campo.

Métrica: Número de decisões tomadas.

QE03: Qual a avaliação da equipe de qualidade quanto a detecção de cenários de limpeza de código?

Fonte: Questionário com equipe de qualidade.

Métrica: muito bom, bom, regular, ruim, muito ruim.

QE04: Com que frequência a equipe de qualidade encontra falhas relacionados à utilização da ferramenta em um determinado intervalo de tempo?

Fonte: Registro de observação em campo.

Métrica: Quantidade de falha / tempo (release, sprint).

Interpretação da métrica: Quanto mais próximo de zero melhor 0=<X

QE05: Qual a quantidade total de falhas encontradas pela equipe de qualidade

relacionadas à utilização da ferramenta?

Fonte: Registro de observação em campo (áudio/vídeo).

Métrica: Quantidade de falhas.

QE06: Qual a proporção do uso da ferramenta para tomada de decisões?

Fonte: Questionário com equipe de qualidade.

Métrica: Número de decisões tomadas / número de vezes que a solução foi usada.

QE07: Qual a quantidade de cenários que foram corrigidos após utilização da solução?

Fonte: Código fonte.

Métrica: Números de cenários corrigidos / número de cenários encontrados.

QE08: Qual o nível de satisfação do uso da solução em comparação à solução anterior?

Fonte: Equipe de qualidade.

Métrica: Muito satisfeito, Satisfeito, Neutro, Insatisfeito, Muito Insatisfeito.

QE09: Qual a taxa de oportunidade de melhoria de código em uma intervalo de tempo (sprint, release)?

Fonte: Código fonte.

Métrica: Taxa de oportunidade de melhoria de código:

$$T_r = \frac{\sum_{i=1}^{n} Ce_i}{\sum_{i=1}^{n} Cl_i}$$

Ce é o total de cenários de limpezas identificados e Cl é o total de classes em um intervalo de tempo (sprint, release).

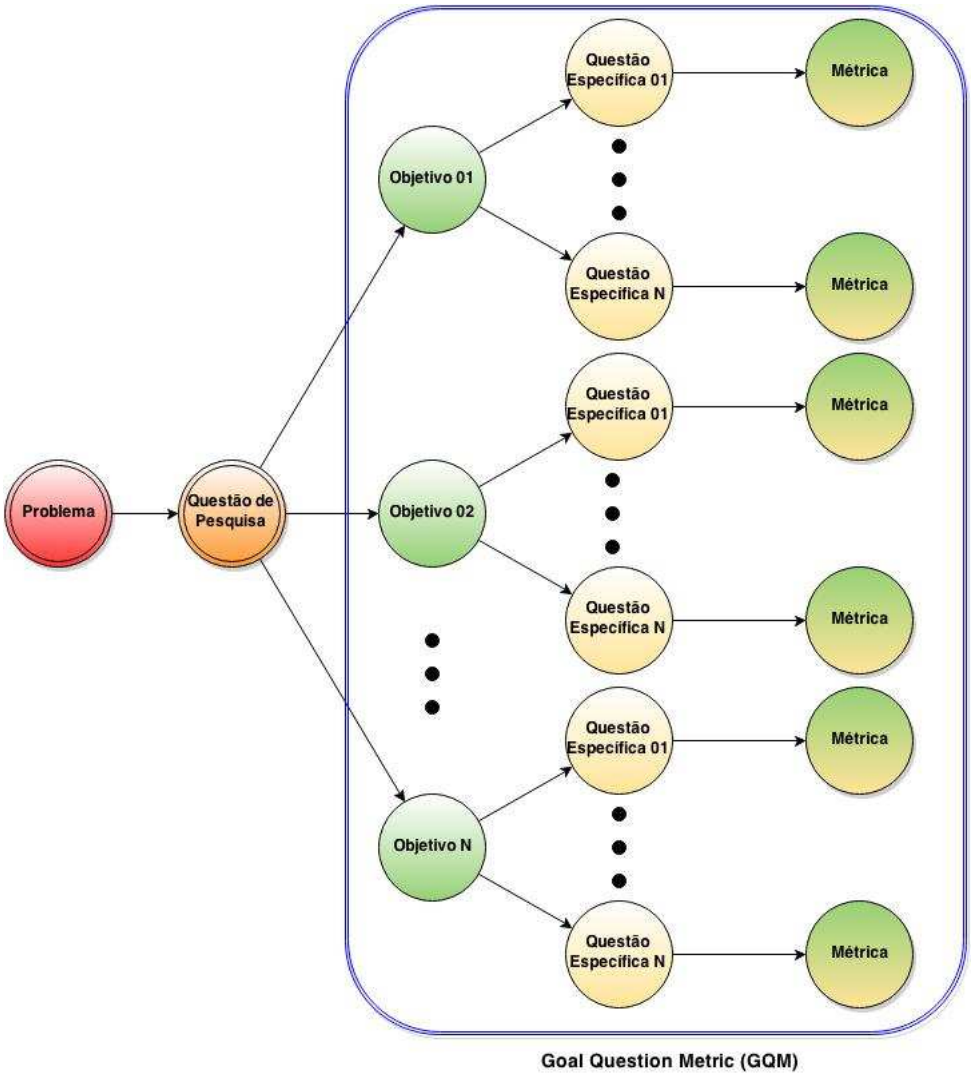

Figura 19 – Estrutura do Estudo de Caso

7.3 Design

Stake (1995) identifica três modalidades de estudos de caso: intrínseco, instrumental e coletivo.

- **Estudo de caso intrínseco:** Constitui o próprio objeto da pesquisa. O que o pesquisador almeja é conhecê-lo em profundidade, sem qualquer preocupação com o desenvolvimento de alguma teoria.
- **Estudo de caso instrumental:** É desenvolvido para auxiliar no conhecimento ou na redefinição de determinado problema. O pesquisador não tem interesse específico no caso, mas reconhece que pode ser útil para alcançar determinados objetivos.
- **Estudo de caso coletivo:** É para estudar características de uma população. Os casos são selecionados porque se acredita que, por meio deles, torna-se possível aprimorar o conhecimento acerca do universo a que pertencem.

Como a busca do entendimento geral sobre o problema de pesquisa será a partir do estudo de um caso particular, o estudo deste trabalho se caracteriza como instrumental. Podemos observar que a definição de estudo de caso exploratório de Yin (2001), onde é elucidada a visão acurada de um caso particular que pode fornecer uma visão geral do problema considerado, que se equivale a definição de estudo de caso instrumental na nomenclatura de (STAKE, 1995).

7.4 Seleção

A organização pública selecionada para este estudo de caso foi a CAIXA. O estudo acontecerá mais especificamente na Centralizadora Nacional de Tecnologia da Informação (CETEC) e na Gerência de Filial de suporte Tecnológico de Brasília (GITECBR).

Atividades atribuídas à CETEC:

- **Suporte para o ambiente descentralizado**
- **Inventário de recurso no ambiente descentralizado**
- **Desenvolvimento e produção de soluções no ambiente descentralizado**
- **Gestão de incidentes e mudanças no ambiente descentralizado**

Atividades atribuídas à GITECBR:

- **Gestão de incidentes tecnológicos de *software* e *hardware***
- **Suporte tecnológico para canais e unidades da CAIXA**
- **Desenvolvimento de soluções e serviços tecnológicos de *software* e *hardware***
- **Comunicação no ambiente descentralizado e externo.**

A homologação dos serviços e a aceitação dos S&SC será facultado à CAIXA, submetendo os programas produzidos pela CONTRATADA a testes em produtos de software especializados para avaliação do desempenho dos mesmos. Atualmente, na CETEC, a solução em homologação, que utiliza as unidades da CAIXA no processo de monitoração de métricas de código-fonte adquiridos por empresas terceirizadas, utiliza a ferramenta SonarQube [1]. Apesar do processo existente estar em fase de homologação, a definição de quais as métricas, indicadores e os seus valores de referência para interpretação, ainda não foi finalizado. Diante disso, as medidas que realmente são utilizadas para o ateste da qualidade interna do produto de *software* entregue estão relacionadas ao número de defeitos e ao índice aceitável de defeitos por ponto de função, conforme pode ser observado na Figura

[1] http://www.sonarqube.org/

20. O número de pontos de defeitos pode gerar multas para a CONTRATADA conforme o índice de defeitos referenciado na tabela 19, porém a multa não exime a CONTRATADA das obrigações de corrigir os erros encontrados, onde ainda as alterações propostas pela CONTRATADA deverão ser efetuadas sem qualquer tipo de ônus financeiro ou a outro projeto para a CAIXA.

Fórmula de Índice de Defeitos

$$Pd = \left[\sum_{1}^{N\ Tipos\ defeitos} (PSE \times Qtd\ Ocorrências) + (PRE \times Qtd\ Reincidência) \right] / Ts$$

Onde:

Pd – Pontos de Defeitos
PSE – Peso da Severidade dos Erros
PRE – Peso de Reincidência de Erros
Ts–Tamanho do Serviço em Ponto de Função

Severidade	Peso
Altíssima	5
Alta	3
Média	2
Baixa	1

Severidade na Reincidência	Peso
Altíssima	7
Alta	4
Média	3
Baixa	1

Figura 20 – Fórmula de Índice de Defeitos

Regras relacionadas à Fórmula de Índice de Defeitos:

- Para arredondamento do valor de "Pd" aplicar-se-á a seguinte regra: se o número constante na segunda casa decimal for superior ou igual a 5, o algarismo da primeira casa decimal será acrescido de 1. Caso contrário, o valor da primeira casa decimal permanece inalterado. (ex: se o resultado do cálculo for igual a 0,18, o valor passará a ser 0,2. Se o resultado do cálculo for igual a 0,13, o valor passará a ser 0,1).
- Caso o índice de defeitos fique superior ao aceitável, que é de 0,2 erro por ponto de função, sensibilizará o indicador de desempenho e aplicará o respectivo percentual de redução, conforme a tabela 19.

Percentual de Redução	Incidência
0.05%	Por erro gerado, a partir do limite tolerável de 0.2 erro por ponto de função.
0.10%	Por erro gerado, quando houve a devolução para acertos e a demanda foi entregue novamente com erros, independentemente do limite tolerável.

Tabela 19 – Multa por defeitos

Para o estudo de caso deste trabalho foi analisado um sistema do portfólio da filial GITECBR desenvolvido na linguagem java. Em cada *release* do projeto a GITECBR recebe um pacote da contratada contendo o código-fonte do sistema. Antes da nova versão do sistema entregue ser implantada em ambiente de homologação de sistemas, ela deve passar pela solução disponibilizada pela CETEC que utiliza o sonarQube. O sonarQube apresenta os defeitos contidos no código-fonte. Com base nos defeitos e no índice aceitável de defeitos, que é de 0,2 erro por ponto de função, o gestor define se a *realease* deve ser rejeitada e se existe alguma multa aplicável conforme a tabela 19.

7.5 Fonte dos Dados Coletados e Método de Coleta

Os dados deste estudo de caso serão coletados através de registros de observações em campo, questionários, entrevistas e resultados gerados pela própria solução de *Data Warehousing* utilizando o código-fonte de um ou mais sistemas desenvolvidos por empresas contratadas pela CAIXA.

Os registros oriundos de observações em campo são gerados a partir da equipe responsável pela tomada de decisões de qualidade e são coletados durante as visitas realizadas em campo. Nessas visitas, a solução proposta será utilizada e qualquer atitude relacionada ao seu uso pela equipe será registrada. Tais registros são tanto do ponto de vista qualitativo quanto do ponto de vista quantitativo.

A adoção de questionários também será utilizada principalmente para dados qualitativos. O questionário fará indagações a respeito da solução de *Data Warehousing* pelo ponto de vista da equipe responsável e dos demais envolvidos na área de qualidade de *software* da CAIXA, no que diz respeito: à tomadas de decisões, detecção de cenários de limpeza de código, falhas relacionados à utilização da ferramenta adotada na solução de DW, ao nível de satisfação no uso da solução e as taxas de oportunidades de melhoria de código.

Dados quantitativos utilizados pela solução de *Data Warehousing* serão coletados a medida em que esta for utilizada pela equipe de qualidade da CAIXA ao longo das *releases* dos projetos adotados.

7.6 Processo de análise dos dados

A análise de dados coletados durante o estudo de caso a ser realizado na CAIXA será feita através de 4 etapas:

- **Categorização:** Organização dos dados em duas categorias - qualitativos e quantitativos. Os dados qualitativos referem-se aos questionários realizados. Os dados

quantitativos, por sua vez, referem-se aos valores numéricos da solução de DW para monitoramento de métricas.

- **Exibição:** Consiste na organização dos dados coletados para serem exibidos através de gráficos, tabelas e texto para poderem ser analisados.
- **Verificação:** Atestar padrões, tendências e aspectos específicos dos significados dos dados. Procurando assim gerar uma discussão e interpretação de cada dado exibido.
- **Conclusão:** Agrupamento dos resultados mais relevantes das discussões e interpretações dos dados anteriormente apresentados.

7.7 Ameaças a validade do estudo de caso

Yin (2001) descreve como principais ameaças relacionadas à validade do estudo de caso as ameaças relacionadas à validade de construção, à validade interna, à validade externa e à confiabilidade. As quatro ameaças definidas por ele, bem como a forma usada nesse trabalho para preveni-las, são descritas da seguinte maneira:

- **Validade de constructo:** A validade de construção está presente na fase de coleta de dados, quando devem ser evidenciadas as múltiplas fontes de evidência e a coleta de um conjunto de métricas para que se possa saber exatamente o que medir e quais dados são relevantes para o estudo, de forma a responder as questões de pesquisa (YIN, 2001). O uso do GQM mitiga a validade de construção, uma vez que estabelece uma lógica entre a questão de pesquisa e as métricas que serão analisadas para respondê-la.
- **Validade interna:** Para Yin (2001), o uso de várias fontes de dados e métodos de coleta permite a triangulação, uma técnica para confirmar se os resultados de diversas fontes e de diversos métodos convergem. Dessa forma é possível aumentar a validade interna do estudo e aumentar a força das conclusões. A triangulação de dados se dará pelas medidas extraídas do código-fonte por meio da solução de *Data Warehousing* (explicada no capítulo 6), pela análise de questionários e pelos dados coletados através de entrevistas.
- **Validade externa:** Yin (2001) recomenda a replicação do estudo em múltiplos casos. Por esse ser um caso único, a generalização não poderá ser alcançada. Este trabalho é o primeiro a verificar a eficácia e eficiência da solução para o estudo de caso na CAIXA, portanto não há como correlacionar os resultados obtidos a nenhum outro estudo.
- **Confiabilidade:** Com relação a confiabilidade, Yin (2001) associa à repetibilidade, desde que seja usada a mesma fonte de dados. Nesse trabalho, o protocolo

de estudo de caso apresentado nessa seção, além da disponibilização da base de dados a serem coletados e analisados, garantem a repetibilidade desse trabalho e, consequentemente, a validade relacionada à confiabilidade.

7.8 Cronograma

As atividades listadas na Tabela 20 servirão de apoio para a execução do estudo de caso:

Atividade	**Mês de Início**	**Mês de Témino**
Estudo De *Plugins* Do *Pentaho BI* para *Dasboard*	Janeiro	Fevereiro
Elaboração de Novos Cenários de Limpeza	Janeiro	Fevereiro
Análise da Taxa	Janeiro	Fevereiro
Revisão e Adequação	Janeiro	Janeiro
Formalização com as Instituições	Fevereiro	Fevereiro
Reunião para Definição Dos Encontros	Fevereiro	Fevereiro
Instalação do Ambiente	Fevereiro	Fevereiro
Coleta de Dados	Março	Junho
Análise dos Dados	Maio	Junho
Redação do Relatório	Maio	Junho
Revisão dos Resultados	Junho	Junho
Revisão Final do Trabalho	Junho	Junho
Entrega Final	Junho	Junho
Elaboração da Apresentação para a Defesa	Junho	Junho
Preparação com o Orientador para a Defesa	Junho	Junho
Apresentação do Trabalho em Banca	Junho	Junho

Tabela 20 – Cronograma

7.9 Considerações finais do capítulo

Esse capítulo teve como objetivo apresentar o protocolo de estudo de caso que será adotado na continuação deste trabalho. No trabalho de conclusão de curso dois serão apresentados os resultados obtidos para as questões específicas apresentadas neste capítulo, a coleta e a análise dos dados coletados assim como a interpretação dos mesmos possibilitando responder a questão de pesquisa abordada.

8 Execução do Estudo de Caso

8.1 Sobre o SonarQube

O Sonar é uma plataforma para gerenciamento de qualidade de código. Esta ferramenta abrange 7 eixos de qualidade de código: Arquitetura e Design, duplicidade, testes unitários, complexidade, erros em potencial, regras de codificação e comentários.

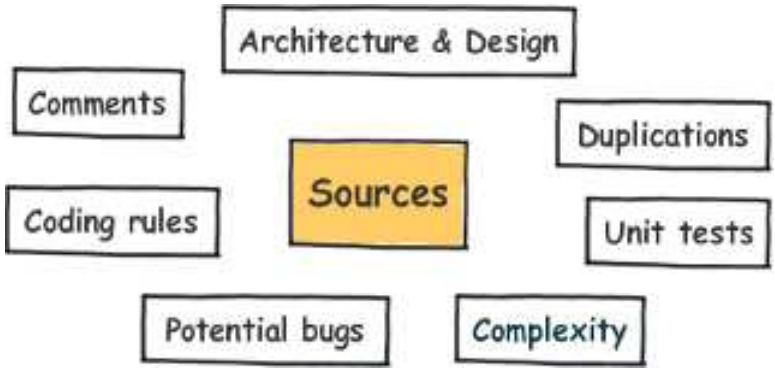

Figura 21 – 7 eixos da qualidade de software SonarQube... (2014)

9 Conclusão

Neste trabalho, foi apresentado o problema do processo de aferição da qualidade interna dos produtos de *software* desenvolvidos por terceirizadas em órgãos públicos brasileiros. Para realizar uma investigação empírica deste problema, com vistas à proposição de uma solução, será feito um estudo do uso de uma solução de *Data Warehouse* para monitoramento de métricas de código-fonte, no contexto da unidade CETEC/CAIXA.

A solução de DW proposta por Rêgo (2014) foi adotada neste trabalho com o objetivo de evidenciar a sua eficácia e eficiência no monitoramento de métricas de código-fonte para assistir ao processo de aferição de qualidade interna dos produtos de software desenvolvido por terceirizadas, do ponto de vista da equipe de qualidade da CAIXA.

Antes da solução de DW ser apresentada, foi realizada uma revisão bibliográfica a respeito de qualidade interna de software, de métricas, de *Data Warehouse* e acerca da definição de estudo de caso. Os conceitos levantados na revisão bibliográfica foram apresentados com o objetivo de facilitar na compreensão da solução adotada. Foi apresentada a plataforma da solução, quer sejam: o ambiente e as ferramentas utilizadas na solução de DW adotada.

Por fim, foi exibido o projeto de pesquisa a ser realizado. Conceitos fundamentais do projeto de pesquisa, como qual o problema a ser resolvido e a questão que o caracteriza, foram identificados e apresentados. Em seguida, utilizou-se da técnica GQM para, através de objetivos, questões específicas e métricas, responder a questão de pesquisa. As etapas pelas quais o estudo de caso atravessará foram descritas e modeladas de forma sequencial, de tal forma a obedecer a revisão bibliográfica sobre o estudo de caso.

A primeira parte do trabalho de conclusão de curso teve como foco o planejamento do estudo de caso. A próxima etapa será responsável pela coleta e análise dos dados a serem obtidos. A coleta e a análise dos dados servirão para evidenciarmos a eficácia e a eficiência do uso de *Data Warehouse* no monitoramento de métricas de código-fonte em um órgão público, com o objetivo de responder a questão de pesquisa definida.

Na segunda parte desse trabalho, também serão definidos mais cenários de limpeza relacionando-os a um conjunto de métricas de código. Adicionalmente, será construído um *dashboard* gerencial visando dar maior visibilidade das medidas analisadas, em particular, procurando verificar como a solução proposta poderia ser utilizada como suporte ao processo de aferição da qualidade interna, hoje realizado na organização analisada.

Referências

BASILI, V. R.; CALDIERA, G.; ROMBACH, H. D. *The Goal Question Metric Approach*. [S.l.]: Encyclopedia of Software Engineering, 1996. Citado na página 73.

BASILI, V. R.; ROMBACH, H. D. *TAME: Integrating Measurement into Software Environments*. 1987. Disponível em: <http://drum.lib.umd.edu//handle/1903/7517>. Citado na página 26.

BECK, K. *Extreme Programming Explained*. [S.l.]: Addison Wesley, 1999. Citado na página 15.

BECK, K. *Implementation Patterns*. 1. ed. [S.l.]: Addison-Wesley Professional, 2007. Citado na página 31.

BRASIL. *Acórdão-381/2011-TCU-Plenário*. [S.l.], 2011. Disponível em: <https://contas.tcu.gov.br/juris/SvlHighLight?key=ACORDAO-LEGADO-89657&texto=2b4e554d41434f5244414f2533413338312b414e442b2b4e55RELACAO-LEGADO;DECISAO-LEGADO;SIDOC;ACORDAO-RELACAO-LEGADO;>. Citado na página 16.

BRERETON, P.; KITCHENHAM, B.; BUDGEN, D. Using a protocol template for case study planning. In: *Proceedings of EASE 2008*. [S.l.]: BCS-eWiC, 2008. Citado na página 69.

BUDD, T. *An introduction to object-oriented programming*. 3rd ed. ed. Boston: Addison-Wesley, 2002. ISBN 0201760312. Citado na página 26.

CDE The Community Dashboard Editors. 2014. <http://www.webdetails.pt/ctools/cde>. Accessed: 2015-02-10. Citado na página 44.

CRUZ, C.; ANDRADE, E.; FIGUEIREDO, R. *Processo de contratação de Serviços de Tecnologia da Informação para Organizações Públicas*. [S.l.]: PBPQ Software, 2011. Citado na página 46.

FENTON, N. E.; PFLEEGER, S. L. *Software Metrics: A Rigorous and Practical Approach*. 2 edition. ed. [S.l.]: Course Technology, 1998. 656 p. Citado 2 vezes nas páginas 25 e 72.

FEW, S. *Information dashboard design: the effective visual communication of data*. 1st ed. ed. Beijing ; Cambride [MA]: O'Reilly, 2006. ISBN 0596100167 9780596100162. Citado 2 vezes nas páginas 42 e 43.

FOWLER, M. *Refactoring: improving the design of existing code*. [S.l.]: Addison-Wesley Professional, 1999. Citado na página 15.

GIL, A. C. *Como elaborar projetos de pesquisa*. [S.l.]: Atlas, 2002. ISBN 9788522431694 8522431698. Citado na página 17.

GOLFARELLI, M. *Data warehouse design: modern principles and methodologies*. New York: McGraw-Hill, 2009. ISBN 9780071610391. Citado 3 vezes nas páginas 54, 55 e 56.

GUTIéRREZ, A.; MAROTTA, A. An overview of data warehouse design approaches and techniques. 2000. Citado na página 53.

HARMAN, M. Why source code analysis and manipulation will always be important. In: IEEE. *Source Code Analysis and Manipulation (SCAM), 2010 10th IEEE Working Conference on*. [S.l.], 2010. Citado na página 25.

HONGLEI, T.; WEI, S.; YANAN, Z. The research on software metrics and software complexity metrics. In: . IEEE, 2009. ISBN 978-1-4244-5422-8, 978-0-7695-3930-0. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5385114>. Citado na página 25.

HOVEMEYER, D.; PUGH, W. Finding bugs is easy. *SIGPLAN Not.*, ACM, New York, NY, USA, v. 39, n. 12, p. 92–106, dez. 2004. ISSN 0362-1340. Disponível em: <http://doi.acm.org/10.1145/1052883.1052895>. Citado na página 40.

IN4. Instrução normativa número 4. 2014. Citado 2 vezes nas páginas 15 e 48.

INMON, W. H. *Building the Data Warehouse*. 3rd. ed. New York, NY, USA: John Wiley & Sons, Inc., 2002. Citado 2 vezes nas páginas 50 e 72.

ISO/IEC 15939. *ISO/IEC 15939: Software Engineering - Software Measurement Process*. [S.l.], 2002. Citado 2 vezes nas páginas 15 e 23.

ISO/IEC 15939. *ISO/IEC 12207: System and Software Engineering - Software Life Cycle Processes*. [S.l.], 2008. Citado 3 vezes nas páginas 23, 46 e 47.

ISO/IEC 9126. *ISO/IEC 9126-1: Software Engineering - Product Quality*. [S.l.], 2001. Citado na página 24.

JELLIFFE, R. Mini-review of java bug finders. O'Reilly Developer Weblogs, mar. 2004. Disponível em: <http://www.oreillynet.com/pub/wlg/4481>. Citado na página 39.

KAN, S. H. *Metrics and models in software quality engineering*. [S.l.]: Addison Wesley, 2002. Citado 3 vezes nas páginas 25, 27 e 72.

KIMBALL, R. *The data warehouse lifecycle toolkit: expert methods for designing, developing, and deploying data warehouses*. [S.l.]: Wiley. com, 1998. Citado na página 55.

KIMBALL, R.; ROSS, M. *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling*. 2nd. ed. New York, NY, USA: John Wiley & Sons, Inc., 2002. ISBN 0471200247, 9780471200246. Citado 7 vezes nas páginas 51, 52, 53, 57, 58, 59 e 72.

LAIRD, M. C. B. L. M. *Software measurement and estimation: A practical approach*. [S.l.]: Wiley-IEEE Computer Society Press, 2006. Citado 2 vezes nas páginas 26 e 27.

LEITE, J. C. Terceirização em informática sob a ótica do prestador de serviços. *Revista de Administração de Empresas*, v. 37, n. 4, 1997. Disponível em: <http://www.scielo.br/pdf/rae/v37n4/a08v37n4>. Citado na página 15.

LOURIDAS, P. Static code analysis. *IEEE Softw.*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 23, n. 4, p. 58–61, jul. 2006. ISSN 0740-7459. Disponível em: <http://dx.doi.org/10.1109/MS.2006.114>. Citado 2 vezes nas páginas 39 e 40.

MACHINI, J. a. et al. *Código Limpo e seu Mapeamento para Métricas de Código-Fonte*. [S.l.]: Universidade de São Paulo, 2010. Citado 7 vezes nas páginas 31, 32, 33, 34, 35, 36 e 72.

MARINESCU, R. Measurement and quality in object-oriented design. In: IEEE. *Software Maintenance, 2005. ICSM'05. Proceedings of the 21st IEEE International Conference on*. [S.l.], 2005. p. 701–704. Citado 2 vezes nas páginas 31 e 71.

MARTIN, R. C. *Clean Code: A Handbook of Agile Software Craftsmanship*. [s.n.], 2008. 464 p. ISBN 9780132350884. Disponível em: <http://portal.acm.org/citation.cfm?id=1388398>. Citado na página 31.

MCCABE, T. J. A Complexity Measure. *IEEE Transactions Software Engineering*, v. 2, n. 4, p. 308–320, December 1976. Citado 2 vezes nas páginas 25 e 72.

MEIRELLES, P. R. M. *Monitoramento de métricas de código-fonte em projetos de software livre*. Tese (Doutorado) — Instituto de Matemática e Estátistica – Universidade de São Paulo (IME/USP), 2013. Citado 6 vezes nas páginas 23, 26, 27, 28, 29 e 71.

MILLS, E. E. Metrics in the software engineering curriculum. *Ann. Softw. Eng.*, J. C. Baltzer AG, Science Publishers, Red Bank, NJ, USA, v. 6, n. 1-4, abr. 1999. ISSN 1022-7091. Citado 2 vezes nas páginas 24 e 25.

NERI, H. R. *Análise, Projeto e Implementação de um Esquema MOLAP de Data Warehouse utilizando SGBD-OR Oracle 8.1*. Universidade Federal da Paraíba - UFPB: [s.n.], 2002. Citado 3 vezes nas páginas 54, 55 e 56.

NOVELLO, T. C. *Uma abordagem de Data Warehouse para análise de processos de desenvolvimento de software*. phdthesis — Pontifícia Universidade Católica do Rio Grande do Sul, 2006. Disponível em: <http://tardis.pucrs.br/dspace/handle/10923/1570>. Citado na página 72.

PUGH, W. Using static analysis to find bugs. *IEEE Softw.*, v. 25, n. 5, 2008. Disponível em: <http://doi.ieeecomputersociety.org/10.1109/MS.2008.130>. Citado na página 39.

RêGO, G. B. Monitoramento de métricas de código-fonte com suporte de um ambiente de data warehousing: um estudo de caso em uma autarquia da administração pública federal. 2014. Disponível em: <http://bdm.unb.br/handle/10483/8069>. Citado 25 vezes nas páginas 15, 16, 17, 28, 29, 30, 32, 33, 34, 35, 36, 37, 50, 56, 57, 58, 59, 60, 61, 62, 63, 67, 68, 71 e 82.

ROCHA, A. B. *Guardando Históricos de Dimensões em Data Warehouses*. Dissertação (Mestrado), Universidade Federal da Paraíba - Centro de Ciências e Tecnologia, 2000. Citado na página 50.

RUIZ, D. D. A. et al. A data warehousing environment to monitor metrics in software development processes. In: *16th International Workshop on Database and Expert Systems Applications (DEXA 2005), 22-26 August 2005, Copenhagen, Denmark*. [S.l.]: IEEE Computer Society, 2005. p. 936–940. ISBN 0-7695-2424-9. Citado na página 72.

SHARMA, N. *Getting started with data warehousing*. [S.l.]: IBM Redbooks, 2011. Citado 5 vezes nas páginas 50, 52, 53, 54 e 72.

SILVA, E.; MENEZES, E. Metodologia da pesquisa e elaboração de dissertação. 2005. Disponível em: <https://projetos.inf.ufsc.br/arquivos/Metodologia_de_pesquisa_e_elaboracao_de_teses_e_dissertacoes_4ed.pdf>. Citado 2 vezes nas páginas 19 e 20.

SOFTEX. *MPS. BR-Guia de Aquisição*. [S.l.], 2013. Disponível em: <http://www.softex.br/wp-content/uploads/2013/07/MPS.BR_Guia_de_Implementacao_SV_Parte_2_20132.pdf>. Citado na página 15.

SOFTEX. MPS BR-guia de implementação – parte 2: Fundamentação para implementação do nível f do mr-mps-sv:2012. 2013. Citado na página 23.

SOMMERVILLE, I. et al. *Engenharia de software*. São Paulo: Pearson Prentice Hall, 2008. ISBN 9788588639287 8588639289. Disponível em: <http://www.worldcat.org/search?qt=worldcat_org_all&q=9788588639287>. Citado na página 39.

SonarQube Kernel Description SonarQube. 2014. <http://www.sonarqube.org/>. Accessed: 2014-10-10. Citado na página 81.

STAKE, R. E. *The art of case study research*. Thousand Oaks: Sage Publications, 1995. ISBN 0803957661. Citado 2 vezes nas páginas 75 e 76.

TIMES, V. C. *Sistemas de DW*. 2012. Universidade Federal de Pernambuco - UFPE. Disponível em: <www.cin.ufpe.br/~if695/bda_dw.pdf>. Citado 2 vezes nas páginas 53 e 54.

WILLCOCKS, L.; LACITY, M. *Global information technology outsourcing*. [S.l.: s.n.], 2001. Citado na página 15.

WOHLIN, C. et al. *Experimentation in software engineering*. [S.l.]: Springer, 2012. Citado na página 20.

YIN, R. *Estudo de caso: planejamento e métodos*. [S.l.]: Bookman, 2001. Citado 5 vezes nas páginas 20, 69, 70, 76 e 79.