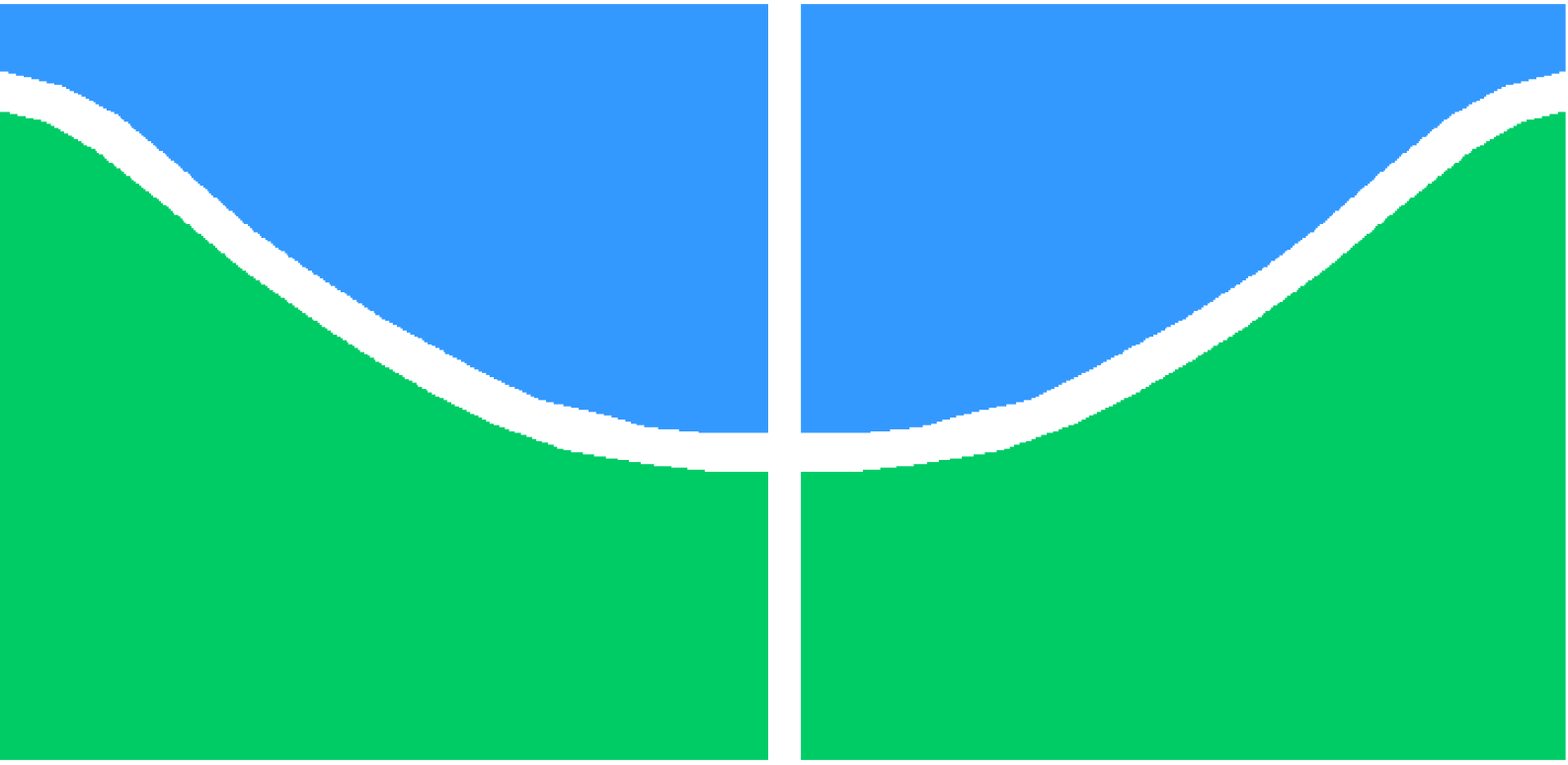

Universidade de Brasília - UnB

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 em uma Autarquia Pública Federal

Autor: Matheus Oliveira Tristão dos Anjos

Orientador: Prof. Msc. Hilmer Rodrigues Neri

Coorientador:

Brasília, DF

2014

Matheus Oliveira Tristão dos Anjos

Estudo da Eficácia e Eficiência do Uso de um Ambiente de *Data Warehousing* para Aferição da Qualidade Interna de Software: Um Estudo de Caso em uma Autarquia Pública 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

Matheus Oliveira Tristão dos Anjos

Estudo da Eficácia e Eficiência do Uso de um Ambiente de *Data Warehousing* para Aferição da Qualidade Interna de Software: Um Estudo de Caso em uma Autarquia Pública Federal/ Matheus Oliveira Tristão dos Anjos. – Brasília, DF, 2014-

52 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 em uma Autarquia Pública Federal

CDU 02:141:005.6

Matheus Oliveira Tristão dos Anjos

Estudo da Eficácia e Eficiência do Uso de um Ambiente de *Data Warehousing* para Aferição da Qualidade Interna de Software: Um Estudo de Caso em uma Autarquia Pública 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, Ainda não se sabe:

Prof. Msc. Hilmer Rodrigues Neri
Orientador

Ainda não se sabe
Convidado 1

Ainda não se sabe
Convidado 2

Brasília, DF
2014

Este trabalho é dedicado aos meus pais - meus maiores exemplos.

Agradecimentos

Agradeço primeiramente ao meu orientador Hilmer Rodrigues Neri pela confiança depositada em mim, por ter me motivado ao longo da graduação a ser um aluno cada vez melhor e pela convivência amigável desde os primeiros semestres. Agradeço aos meus amigos e colegas Nilton Araruna e Pedro Tomioka pela colaboração e ajuda não só durante a execução desse trabalho mas também em toda a graduação. A eles agradeço também pelo sincero sentimento de amizade.

Agradeço ao meu pai e a sua esposa Alice pela paciência e motivação durante a execução desse trabalho, assim como a minha mãe e seu marido João David pela força, apoio, entusiasmo e carinho de sempre. Ainda no desejo de agradecer meus pais, agradeço por todo o conforto, ensinamentos e amor dedicados a mim nos momentos mais difíceis da graduação.

Agradeço aos meus avós e tios pelo apoio de sempre, tendo sido fundamentais nos momentos de conquista e também de dificuldade. A eles externo meus sentimentos de carinho e gratidão.

Agradeço a Louize Helena por todas as alegrias que me proporciona e por transformar qualquer momento de preocupação e desequilíbrio em um momento belo e de otimismo, tornando não só a minha graduação mas qualquer outra coisa que eu faça um lugar mais bonito.

Agradeço ao Serviço de Integração e Métricas do Tribunal de Contas da União pela oportunidade de estágio que me foi dada, me fazendo amadurecer e me tornando um profissional completamente diferente do que era antes. Mais especificamente, gostaria de agradecer meus supervisores Edmilson Faria Rodrigues e Marcus Vinicius Borela pela paciência e bom humor quando precisaram guiar meus passos no estágio.

Agradeço também aos meus amigos e colegas Carlos Filipe Bezerra, Pedro Henrique Potiguara, Gabriela Matias Navarro, Guilherme de Lima, Tiago Pereira, Guilherme Baufaker, Rafael Ferreira, Ramon Cruz, Guilherme Calixto, Fernando Paixão, Ilton Garcia, Rodrigo Rincon, Thabata Granja, Vitor Magalini, Vitor Gonçalves, Wagner Talarico e Mairon Cruvinel.

"Toda ação humana, quer se torne positiva ou negativa, precisa depender de motivação."
(Albert Einstein)

Resumo

Monitorar métricas de código fonte de um software significa monitorar sua qualidade interna. Embora seja possível extrair valores de métricas de código com facilidade através de ferramentas já existentes, a decisão sobre o que fazer com os dados extraídos ainda esbarra na dificuldade relacionada à visualização e interpretação dos dados. Nesse contexto, este trabalho busca analisar a eficácia e eficiência do uso de um ambiente de *Data Warehousing* para facilitar a interpretação das métricas de código fonte, associando-as a cenários de limpeza com o objetivo de apoiar as tomadas de decisão a respeito de mudanças no código. Este trabalho apresenta as fundamentações teóricas necessárias para o entendimento dessa solução bem como elementos que dizem respeito a sua arquitetura e requisitos de negócio. Para realizar a pesquisa sobre sua eficácia e eficiência, foi elaborada uma investigação empírica através da técnica do estudo de caso, que visa responder questões qualitativas e quantitativas a respeito do uso desse ambiente em um órgão público federal.

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

Abstract

Monitor metrics of source code of a software means to monitor its internal quality. Although it is possible to extract values from code metrics with ease through existing tools, the decision about what to do with the extracted data still faces the difficulty related to the visualization and interpretation of data. In this context, this paper seeks to analyze th effectiveness and efficiency of the use of and environment of *Data Warehousing* to facilitate the interpretation of source code metrics, linking them to cleasing scenarios with the aim of supporting decision-making at regarding changes in the code. This paper presents the theoretical framework necessary for understanding this solution as well as elements that relate to their architecture and business requirements. To conduct research on their effectiveness and efficiency, an empirical research has been prepared by the technique of case study that aims to answer qualitative and quantitative questions regarding the use of this environment on a federal public agency.

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

Lista de ilustrações

Figura 1 – Estrutura do estudo de caso 17
Figura 2 – Modelo de Qualidade do Produto adaptado da ISO/IEC 25023 (2011) . 21
Figura 3 – Arquitetura de um ambiente de Data Warehousing 33
Figura 4 – Esquema estrela extraído de Times (2012) 34
Figura 5 – Cubo de dados com o fato Venda e dimensões Produto, Funcionário e Tempo 35
Figura 6 – Diferença entre o fluxo seguido pelo Drill Up e pelo Drill Down, extraída de (SHARMA, 2011) 36
Figura 7 – Arquitetura do ambiente de *Data Warehousing* para métricas de código 37
Figura 8 – Projeto físico do *Data Warehouse* extraído de Rêgo (2014) 41
Figura 9 – Projeto físico do *Data Warehouse* extraído de Rêgo (2014) 42
Figura 10 – Estrutura do estudo de caso 44

Lista de tabelas

Tabela 1 – Percentis para métrica RFC em projetos Java extraídos de Meirelles (2013) 24
Tabela 2 – Nome dos Intervalos de Frequência extraídos de Rêgo (2014) 24
Tabela 3 – Configurações para os Intervalos das Métricas para Java extraídas de Rêgo (2014) 25
Tabela 4 – Conceitos de Limpeza levantados por Almeida e Miranda (2010) extraídos de Rêgo (2014) 27
Tabela 5 – Cenários de Limpeza extraídos de Rêgo (2014) 29
Tabela 6 – Diferenças entre OLTP e OLAP extraídas de Neri (2002) 35
Tabela 7 – Fatos e dimensões identificadas por Rêgo (2014) 39
Tabela 8 – Tabelas fatos e tabelas dimensões elaboradas por Rêgo (2014) 40

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*
CSV	*Comma-Separated Values*
DER	Diagrama Entidade Relacionamento
DIT	*Depth of Inheritance Tree*
DW	*Data Warehouse*
ETL	*Extraction-Transformation-Load*
FTP	*File Transfer Protocol*
GQM	*Goal-Question-Metric*
IEC	*International Electrotechnical Commission*
IPHAN	Instituto do Patrimônio Histórico e Artístico Nacional
ISO	*International Organization for Standardization*
JSON	*JavaScript Object Notation*
LCOM4	*Lack of Cohesion in Methods*
LOC	*Lines of Code*
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*
SGBD	Sistema de Gerenciamento de Bancos de Dados
SICG	Sistema Integrado de Conhecimento e Gestão
XML	*Extensible Markup Language*
YAML	*YAML Ain't Markup Language*

Sumário

1 INTRODUÇÃO 15
1.1 Contexto 15
1.2 Justificativa 15
1.3 Problema 15
1.4 Objetivos 16
1.5 Metodologia de pesquisa 16
1.6 Organização do Trabalho 17

2 MÉTRICAS DE SOFTWARE 19
2.1 Processo de Medição 19
2.2 Definição das Métricas de Software 20
2.3 Métricas de Código Fonte 21
2.3.1 Métricas de Tamanho e Complexidade 21
2.3.2 Métricas de Orientação a Objetos 22
2.4 Configurações de Qualidade para Métricas de Código Fonte 23
2.5 Cenários de Limpeza 26
2.6 Considerações Finais do Capítulo 30

3 *DATA WAREHOUSE* 31
3.1 Definição 31
3.2 *Extraction-Transformation-Load* 31
3.3 Modelagem Dimensional 33
3.3.1 OLAP *(On-Line Analytic Processing)* 34
3.4 Ambiente de *Data Warehousing* para Métricas de Código-Fonte 36
3.5 Considerações finais do capítulo 42

4 PROJETO DE ESTUDO DE CASO 43
4.1 Definição sobre estudo de caso 43
4.1.1 Problema 44
4.1.2 Questão de Pesquisa 44
4.2 *Background* 46
4.3 Seleção 47
4.4 Fonte dos dados coletados e método de coleta 47
4.5 Ameaças a validade do estudo de caso 47
4.6 Processo de análise dos dados 48
4.7 Considerações finais do capítulo 49

5 CONCLUSÃO 50

Referências 51

1 Introdução

1.1 Contexto

Medir a qualidade do código-fonte de um software é um processo fundamental no seu desenvolvimento, pois daí surgem indicadores sobre os efeitos que uma alteração no código irá causar ou sobre os efeitos gerados na qualidade do software após a adesão de uma nova prática na equipe de desenvolvimento (FENTON; PFLEEGER, 1998).

Através do processo de medição da qualidade interna do software é possível não só entender e controlar o que está se passando no seu desenvolvimento, como também ser encorajado a tomar decisões que visem a sua melhoria (FENTON; PFLEEGER, 1998). Como a qualidade do código fonte está relacionada à qualidade interna do produto de software (ISO/IEC 25023, 2011), sua medição, e consequentemente as melhorias implantadas nele melhoram a qualidade do produto como um todo.

1.2 Justificativa

Buscando facilitar a interpretação das métricas de código fonte, bem como apoiar as decisões de refatoração a serem tomadas, Rêgo (2014) elaborou um ambiente de *Data Warehouse* para armazenamento de métricas, sobre o qual esse trabalho irá analisar sua eficácia e eficiência no monitoramento da qualidade interna do software, sendo essa sua maior contribuição. Essa análise poderá ajudar equipes de desenvolvimento de software na escolha de uma maneira de aferir a qualidade interna do seu produto e tomar as decisões corretas sobre o que refatorar.

1.3 Problema

ESCREVER PROBLEMA

Baseando-se no problema descrito, foi criada a seguinte questão geral de pesquisa:

O uso de Data Warehousing é eficaz e eficiente no monitoramento de métricas de código fonte, facilitando a interpretação dos resultados da análise estática e apoiando decisões de refatoração?

1.4 Objetivos

O objetivo geral desse trabalho consiste na análise da eficácia e eficiência do uso de *Data Warehousing* no monitoramento de métricas de código fonte. Como esse trabalho é dividido em duas partes, seus objetivos também foram divididos dessa forma, sendo os objetivos da primeira parte:

- Levantar fundamentação teórica que servirá de base para todo o trabalho.
- Apresentar solução desenvolvida por Rêgo (2014) para monitoramento de métricas através de *Data Warehousing*.
- Elaborar o projeto do estudo de caso.

A segunda parte desse trabalho, a ser desenvolvida no primeiro semestre de 2015, terá os seguintes objetivos:

- Coletar os dados que surgirão como respostas das questões específicas elaboradas para o estudo de caso, como por exemplo o nível de satisfação quanto ao uso da solução ou qual a taxa de oportunidade de melhoria de código em um determinado intervalo de tempo.
- Realizar análise dos dados coletados.
- Relatar resultados obtidos.

1.5 Metodologia de pesquisa

A metodologia adotada:

Segundo Wohlin et al. (2012) a condução de um estudo de caso deve obedecer os cinco passos listados de forma sequencial na figura 1:

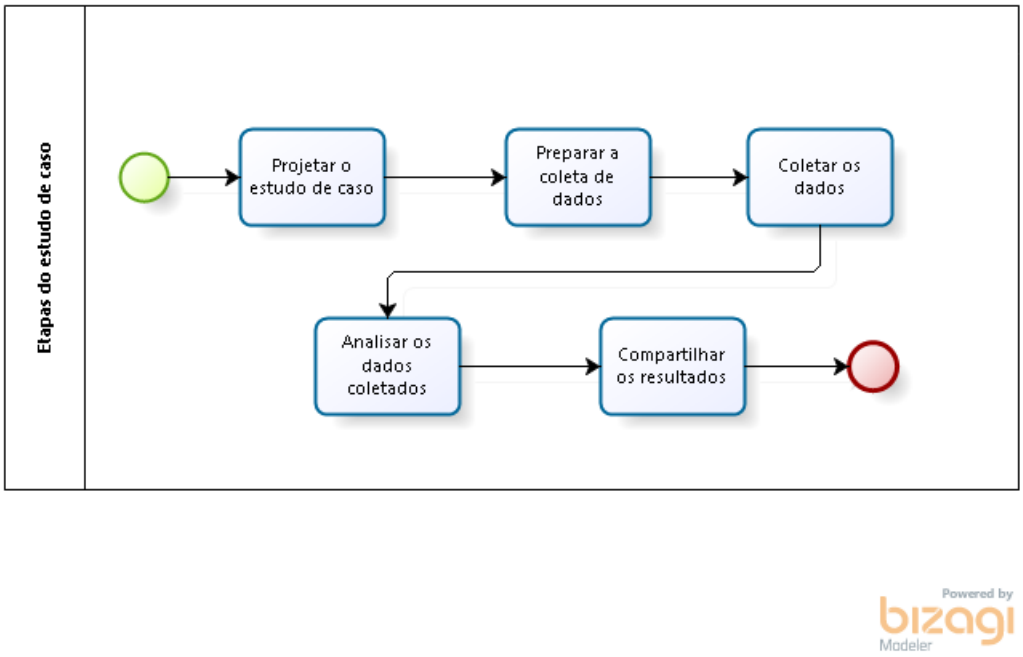

Figura 1 – Estrutura do estudo de caso

Projetar o estudo de caso e preparar a coleta de dados são etapas que consistem em uma macro atividade chamada de planejamento do estudo de caso. Nesse planejamento serão identificados elementos como o problema a ser resolvido e a questão de pesquisa a ser respondida, de modo que o protocolo de estudo de caso possa ser elaborado já determinando qual o método e a fonte da coleta dos dados. Coletar dados é uma atividade a ser executada em seguida, feita através de pesquisas bibliográficas e documentais além dos resultados extraídos da própria solução para monitoramento de código fonte a ser analisada durante o estudo de caso. Analisar os dados coletados diz respeito à interpretação dos dados coletados compreendendo não só análises quantitativas como também qualitativas. A etapa chamada de compartilhar os resultados, por sua vez, indica que os resultados coletados e interpretados são expostos nesse trabalho a quem desejar fazer sua leitura.

1.6 Organização do Trabalho

Esse trabalho está dividido em 5 capítulos:

- **Capítulo 1 - Introdução:** Esse capítulo tem como objetivo apresentar o contexto que esse trabalho está inserido, o problema sobre o qual ele buscará resolver, qual a justificativa e os objetivos da sua realização e como essa pequisa foi elaborada.
- **Capítulo 2 - Métricas de Software:** Capítulo responsável pela explicação teórica a respeito do que são métricas de código e como elas foram utilizadas no desenvol-

vimento da solução que esse trabalho busca analisar.

- **Capítulo 3 - Data Warehouse:** Nesse capítulo serão apresentados conceitos teóricos sobre *Data Warehousing*, assim como a maneira como foi desenvolvido o ambiente de *Data Warehouse* para armazenamento de métricas de código fonte.

- **Capítulo 4 - Projeto de estudo de caso:** Será apresentada a estratégia de pesquisa adotada durante o trabalho, buscando elaborar um protocolo para o estudo de caso que será realizado. Elementos de pesquisa como o problema a ser resolvido, os objetivos a serem alcançados no estudo de caso e quais os métodos de coleta e análise dos dados serão identificados e explicados.

- **Capítulo 5 - Conclusão:** Além das considerações finais dessa primeira parte do trabalho, serão descritos objetivos para a segunda parte a ser realizada ao término desta.

2 Métricas de Software

Esse capítulo será 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. A explicação teórica envolve desde o entendimento do processo de medição descrito pela ISO/IEC 15939 (2002) até a explicação sobre intervalos qualitativos para valores de cada métrica de código fonte propostos por Meirelles (2013). Por fim, serão apresentados os cenários de limpeza de código utilizados nesse trabalho para monitoramento da qualidade do código fonte.

2.1 Processo de Medição

A ISO/IEC 15939 (2002) define medição como a união de operações cujo objetivo é atribuir um valor a uma métrica. Graças ao processo de medição é possível entender e controlar o que está acontecendo durante o processo de desenvolvimento e manutenção de um sistema Fenton e Pfleeger (1998). Ainda segundo a ISO/IEC 15939 (2002), o processo de medição é a chave primária para a gerência de um software e suas atividades no seu ciclo de vida, além disso, um processo de melhoria contínua requer mudanças evolutivas e mudanças evolutivas requerem um processo de medição. Complementando o conceito levantado anteriormente, é possível afirmar de acordo com a ISO/IEC 9126 (2001) que a medição é a utilização de uma métrica para atribuir um valor, que pode ser um número ou uma categoria, obtido a partir de uma escala a um atributo de uma entidade. A escala, citada anteriormente, pode ser definida como um conjunto de categorias para as quais os atributos estão mapeados, de modo que um atributo de medição está associado a uma escala ISO/IEC 15939 (2002). Essas escalas podem ser divididas em:

- **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 a medida com escala do tipo intervalo, porém a proporção possui significado. (MEIRELLES, 2013)

A ISO/IEC 15939 (2002) 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

Fenton e Pfleeger (1998), mostraram que o termo métricas de software abrange muitas atividades, as quais estão envolvidas em um certo grau de medição de um software, como por exemplo estimativa de custo, estimativa de esforço e capacidade de reaproveitamento de elementos do software. Nesse contexto ISO/IEC 9126 (2001) categoriza as seguintes métricas de acordo com os diferentes tipos de medição:

- **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.

A figura 2 reflete como as métricas influenciam nos contextos em que elas estão envolvidas, seja em relação ao software propriamente dito (tanto internamente quanto externamente) ou ao efeito produzido pelo uso de software:

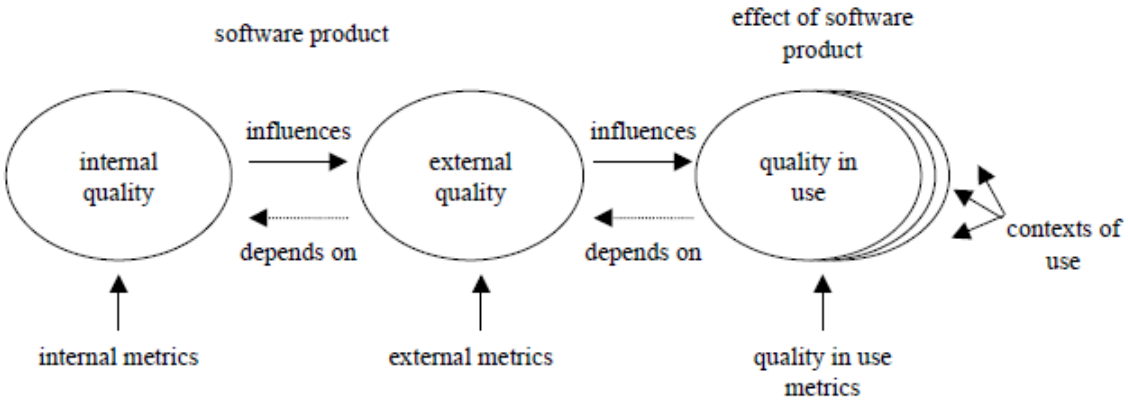

Figura 2 – Modelo de Qualidade do Produto adaptado da ISO/IEC 25023 (2011)

2.3 Métricas de Código Fonte

Serão utilizadas nesse trabalho de conclusão de curso métricas de código fonte, que segundo Meirelles (2013) são métricas do tipo objetiva calculadas a partir da análise estática do código fonte de um software. As métricas de código fonte serão divididas em duas categorias, seguindo a categorização adotada por Rêgo (2014): Métricas de tamanho e complexidade e métricas de orientação a objetos.

2.3.1 Métricas de Tamanho e Complexidade

O tamanho do código-fonte de um sistema foi um dos primeiros conceitos mensuráveis de software, uma vez que softwares podiam ocupar espaço tanto em forma de cartão perfurado quanto em forma de papel quando o código era impresso. Na programação em Assembler, por exemplo, uma linha física de código era o mesmo que uma instrução, logo, quanto maior o tamanho do código, maior era sua complexidade (KAN, 2002). 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

O surgimento da programação orientada a objetos representou uma importante mudança na estratégia de desenvolvimento, focalizando a atenção para conceitos mais próximos ao negócio modelado. (GILMORE, 2008)

Métricas de orientação a objetos foram adotadas devido à grande utilização desse paradigma no desenvolvimento de software. Serão adotadas as seguintes métricas já selecionadas por Rêgo (2014):

- **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

Em sua tese de doutorado, Meirelles (2013) buscou responder as seguintes questões de pesquisa:

- **QP1 -** Métricas de código-fonte podem influir na atratividade de projetos de software livre?

- **QP1 -** Quais métricas devem ser controladas ao longo do tempo?

- **QP3 -** As métricas de código-fonte melhoram com o amadurecimento do projeto?

Para responder essas questões, sua pesquisa concentrou-se em alguns objetivos tecnológicos e científicos, fazendo uso da técnica estatística descritiva percentil para identificação das distribuições dos valores de métricas em 38 projetos de software livre, observando os valores frequentes dessas métricas de modo a servirem de referência para projetos futuros.

O percentil são pontos estimativos de uma distribuição de frequência que determinam a porcentagem de elementos que se encontram abaixo deles. Por exemplo, quando se diz que o valor 59,0 da métrica rfc do projeto **Open JDK8** está no percentil 90, significa dizer que 90% dos valores identificados para essa métrica estão abaixo de 59,0.

A tabela 1 abaixo pôde ser criada graças ao uso da técnica estatística citada:

	Mín	**1%**	**5%**	**10%**	**25%**	**50%**	**75%**	**90%**	**95%**	**99%**	**Máx**
Eclipse	1,0	1,0	1,0	1,0	4,0	11,0	28,0	62,0	99,0	221,0	3024,0
Open JDK8	1,0	1,0	1,0	1,0	3,0	9,0	26,0	59,0	102,0	264,0	1603,0
Ant	1,0	1,0	1,0	2,0	5,0	14,0	34,0	72,0	111,0	209,0	405,0
Checkstyle	1,0	1,0	1,0	1,0	2,0	6,0	16,0	31,0	42,0	80,0	270,0
Eclipse Metrics	1,0	1,0	1,0	2,0	4,0	7,0	19,0	37,0	57,0	89,0	112,0
Findbugs	1,0	1,0	1,0	1,0	2,0	6,0	17,0	43,0	74,0	180,0	703,0
GWT	1,0	1,0	1,0	1,0	2,0	7,0	20,0	39,0	65,0	169,0	1088,0
Hudson	1,0	1,0	1,0	1,0	3,0	6,0	14,0	27,0	45,0	106,0	292,0
JBoss	1,0	1,0	1,0	1,0	3,0	7,0	15,0	31,0	49,0	125,0	543,0
Kalibro	1,0	1,0	1,0	2,0	4,0	8,0	15,0	29,0	39,0	58,0	84,0
Log4J	1,0	1,0	1,0	2,0	4,0	8,0	23,0	52,0	85,0	193,0	419,0
Netbeans	1,0	1,0	1,0	1,0	3,0	9,0	21,0	46,0	72,0	164,0	2006,0
Spring	1,0	1,0	1,0	1,0	2,0	6,0	17,0	41,0	66,0	170,0	644,0
Tomcat	1,0	1,0	1,0	2,0	4,0	11,0	30,0	74,0	130,0	275,0	1215,0

Tabela 1 – Percentis para métrica RFC 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 1, por exemplo, foram observados no projeto **Open JDK8** valores de 0 a 9 como muito frequentes, de 10 a 26 como frequente, de 27 a 59 como pouco frequente e acima de 59, que representa apenas 10% do código-fonte do projeto, como não frequente (MEIRELLES, 2013). A tabela 2 foi extraída do trabalho de conclusão de curso de Rêgo (2014) para que fosse criada uma relação entre o intervalo de frequência e o intervalo qualitativo de uma métrica, afim de facilitar sua interpretação:

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

Tabela 2 – Nome dos Intervalos de Frequência extraídos de Rêgo (2014)

Meirelles (2013) destaca na avaliação dos resultados a maneira como o **Open JDK8** demonstrou um equilíbrio no valor das métricas em relação aos demais projetos Java, de modo que seus valores são frequentemente usados como referência na interpretação dos valores das métricas. Se de um lado o **Open JDK8** demonstrou os menores valores percentis, os valores mais altos foram identificados no **Tomcat**. Rêgo (2014) considerou então os dois cenários para que as referências de valores para as métricas fossem criadas. O resultado dessa análise gerou em seu trabalho a tabela 3:

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

2.5 Cenários de Limpeza

Em seu livro Implementation Patterns, Beck (2007) destaca três valores que um código limpo precisa ter: Comunicabilidade , simplicidade e flexibilidade.

- **Comunicabilidade:** 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:** Eliminar o excesso de complexidade faz com que aqueles que estejam lendo o código a entendê-lo mais rapidamente. O excesso de complexidade faz com que seja maior a probabilidade de erro e com que seja mais difícil fazer uma manutenção no futuro.Buscar simplicidade é também buscar inovação: *Junit* é muito mais simples que muitas ferramentas de teste que ele substituiu.

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

Buscando levantar conceitos que fizessem com que um código atendesse os valores citados acima, tornando-se assim um código limpo, Almeida e Miranda (2010) levantaram em seu trabalho conceitos de limpeza, evidenciando as contribuições e consequências que o uso da técnica pode causar no código. Serão citadas na tabela 4 técnicas levantadas por Almeida e Miranda (2010) que ganharam destaque no trabalho de Rêgo (2014):

Conceito de Limpeza	Descrição	Consequências de Aplicação
Composição de Métodos	Compor os métodos em chamadas para outros rigorosamente no mesmo nível de abstração abaixo.	• Menos Operações por Método • Mais Parâmetros de Classe • Mais Métodos na Classe
Evitar Estruturas Encadeadas	Utilizar a composição de métodos para minimizar a quantidade de estruturas encadeadas em cada método (if, else).	• Menos Estruturas encadeadas por método (if e else) • Benefícios do Uso de Composição de Métodos
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.	• 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.	• 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.	• Menos Parâmetros passados pela Classe • Possível diminuição na coesão
Uso Excessivo de Herança	Localizar uso excessivo de herança e transformá-lo em agregação simples.	• 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.	• Maior Encapsulamento de Parâmetros

Tabela 4 – Conceitos de Limpeza levantados por Almeida e Miranda (2010) extraídos de Rêgo (2014)

Após o levantamento dos conceitos de limpeza, Almeida e Miranda (2010) criaram um mapeamento os relacionando com métricas de código, definindo cenários de limpeza. O objetivo, como ressaltado pelo autor, não era classificar um código como limpo ou não, mas sim facilitar melhorias de implementação através da aproximação dos valores das métricas com os esperados nos contextos de interpretação.

Aproveitando inicialmente os cenários **Classe pouco coesa** e **Interface dos métodos** extraídos de Almeida e Miranda (2010), Rêgo (2014) elaborou mais alguns cenários de limpeza, utilizando como referência a configuracão do **Open JDK8**, considerando como valores altos os valores obtidos pelos intervalos Regular e Ruim para esse sistema. O resultado dessa atividade pode ser visto na tabela 5:

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 Parâmetro 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 Estrutura 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 5 – Cenários de Limpeza extraídos de Rêgo (2014)

2.6 Considerações Finais do Capítulo

Esse capítulo apresentou a fundamentação teórica sobre aquilo que é medido e monitorado pela solução proposta. Além de uma definição sobre o que são métricas de código fonte e quais são seus intervalos qualitativos, também foi apresentada nesse capítulo a maneira como elas foram relacionadas a cenários de limpeza. O próximo capítulo será responsável por apresentar a teoria acerca de *Data Warehousing* e como seu uso pode servir no monitoramento das métricas apresentadas nesse capítulo.

3 *Data Warehouse*

Neste capítulo será apresentada a maneira como foi desenvolvido o ambiente de *Data Warehouse* para armazenamento de métricas de código fonte, sobre o qual esse trabalho busca analisar a eficácia e eficiência no monitoramento de métricas. Serão apresentados anteriormente conceitos teóricos sobre *Data Warehouse*, para então relacionar seu uso dentro do contexto de métricas.

3.1 Definição

Data Warehouse é uma base de dados que armazena suas informações de maneira orientada a satisfazer solicitações de tomadas de decisão (CHAUDHURI; DAYAL, 1997). A diferença entre um típico banco de dados transacional e um *Data Warehouse*, porém, consiste na maneira como esses dados são armazenados. Em vez de existirem múltiplos ambientes de decisão operando de forma independente, o que com frequência traz informações conflituosas, um *Data Warehouse* unifica as fontes de informações relevantes, de maneira que a integridade qualidade dos dados são garantidas. (SHARMA, 2011). Dessa forma,Chaudhuri e Dayal (1997) afirma que o ambiente de *Data Warehousing* possibilita que seu usuário realize buscas complexas de maneira mais amigável diretamente em um só ambiente, em vez de acessar informações através de relatórios gerados por especialistas.

Inmon (2002) descreve que o *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, como por exemplo as vendas de uma loja. O fato do ambiente ser integrado remete ao fato dele ser alimentado com dados que têm como origem de 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.

3.2 *Extraction-Transformation-Load*

Para alcançar as características descritas na definição, o ambiente de Data Warehousing segue o processo que consiste na extração, transformação e carga dos dados, conhecido como *Extraction-Transformation-Load* (ETL). Cada um dos passos recebe a seguinte descrição:

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

A figura 3 descreve a arquitetura de um ambiente de *Data Warehousing*, envolvendo os três processos citados anteriormente:

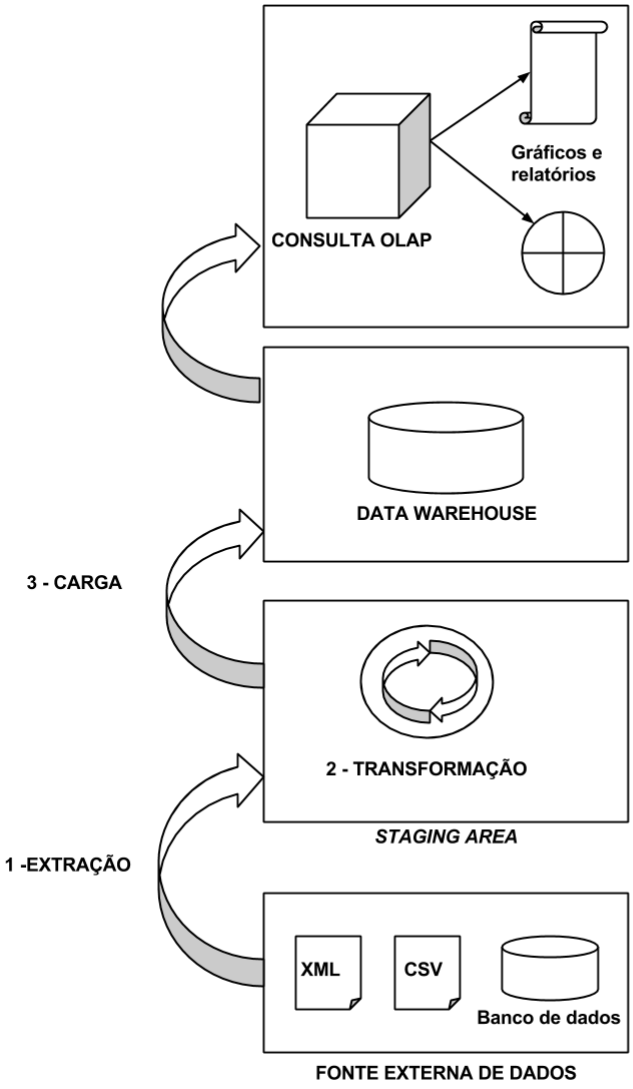

Figura 3 – Arquitetura de um ambiente de Data Warehousing

3.3 Modelagem Dimensional

Kimball e Ross (2002) afirma que a habilidade de visualizar algo tão abstrato como um conjunto de dados de maneira concreta e tangível é o segredo da compreensibilidade, de modo que um modelo de dados que se inicia de forma simples tende a ser simples até o final da modelagem, ao contrário de um modelo que já se inicia de forma complicada. Nesse contexto, o modelo dimensional difere em muitos aspectos do modelo normalizado em sua terceira forma normal, também conhecido como modelo entidade-relacionamento. O modelo normalizado contém seus dados divididos em muitas entidades, cada qual identificada como uma tabela, buscando assim evitar redundância entre os dados, sendo eles armazenados em tempo real na medida que forem atualizados. O problema associado a essa solução é a tamanha complexidade adquirida pelos modelos, uma vez que são criadas um número grande de tabelas dificultando assim sua navegação. Em um sentido oposto, a modelagem dimensional resolve esse problema associado à complexidade, uma

vez que, mesmo possuindo as mesmas informações que um modelo normalizado, elas estão modeladas de forma que estejam em sintonia com o entendimento do usuário e ao alto desempenho de consultas.

Um modelo dimensional é composto por tabelas fatos e tabelas dimensões, que quando juntas formam o esquema estrela. A tabela fato é a tabela primária no modelo dimensional. O termo *fato* está associado à maneira como ela representa uma medida de negócio (KIMBALL; ROSS, 2002). Já a tabela dimensão contém descrições textuais dos negócios envolvidos, o que a torna a chave para que o modelo seja utilizável e de fácil entendimento. Kimball e Ross (2002) faz uma relação direta entre a qualidade do Data Warehouse como um todo e a qualidade e profundidade dos atributos das tabelas dimensão. O esquema estrela, já definido como a união entre tabelas fato e dimensão, pode ser representado da forma como o exemplo da figura 4 descreve:

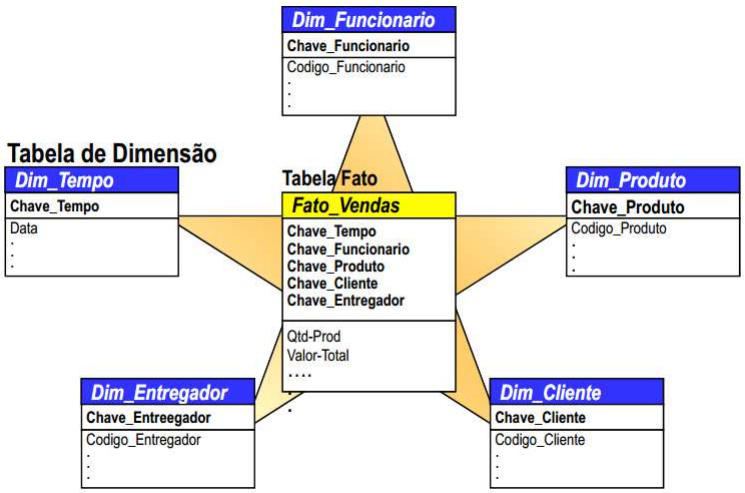

Figura 4 – Esquema estrela extraído de Times (2012)

3.3.1 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). A consulta OLAP difere da consulta do tipo *On-Line Transaction Processing* (OLTP) pelo fato dos seus dados terem passado por um processo de ETL, de modo que sua performance foi melhorada para uma análise mais fácil e rápida, enquanto na consulta do tipo OLTP o sistema foi modelo de modo a capacitar inserções, atualizações e remoções de dados obedecendo regras de normalização. A tabela 6 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 6 – Diferenças entre OLTP e OLAP extraídas de Neri (2002)

A modelagem multidimensional adotada pelo OLAP é associada de maneira metafórica na literatura a um cubo de dados, cujas arestas definem as dimensões dos dados e as células do cubo contém valores de medida (KIMBALL; ROSS, 2002). Os cubos de dados têm um foco nas necessidades de negócio e podem ser exemplificados como na figura 5:

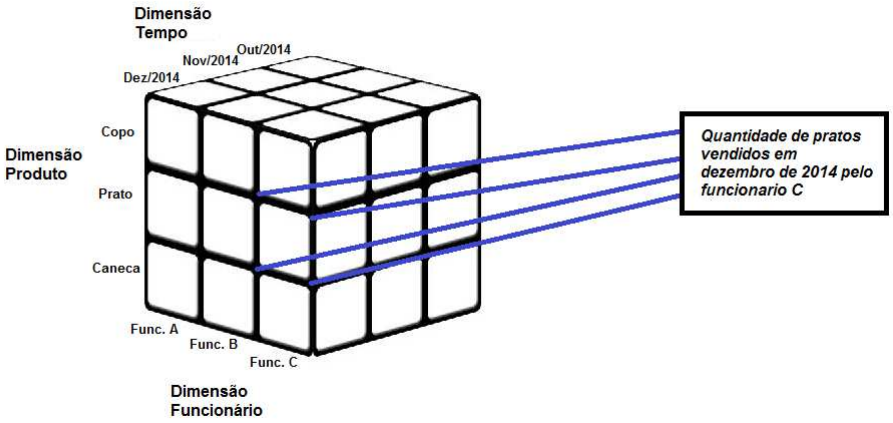

Figura 5 – Cubo de dados com o fato Venda e dimensões Produto, Funcionário e Tempo

A figura 5 reflete uma análise em um cubo de dados para o esquema estrela da figura 4. Essas análises podem ser divididas em alguns tipos diferentes, entre eles:

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

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

A figura 6, extraída de Sharma (2011), mostra o fluxo percorrido pelo *Drill Down* e pelo *Drill Up* nas consultas OLAP:

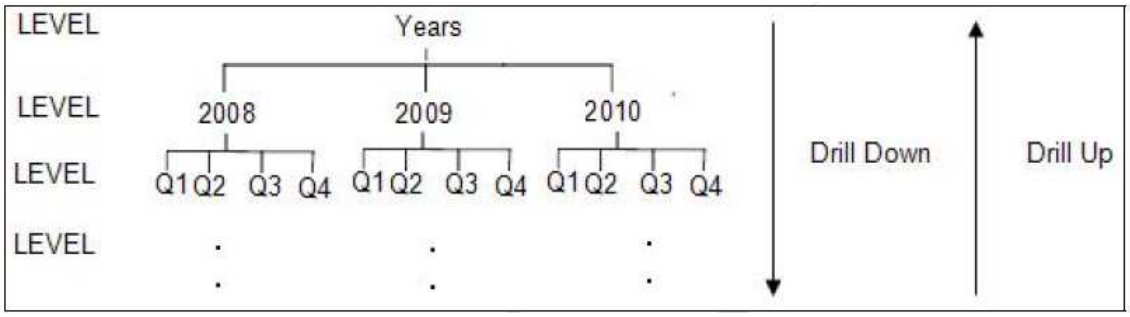

Figura 6 – Diferença entre o fluxo seguido pelo Drill Up e pelo Drill Down, extraída de (SHARMA, 2011)

3.4 Ambiente de *Data Warehousing* para Métricas de Código-Fonte

A figura 3 descreve a arquitetura de um ambiente de *Data Warehousing*. O nível mais baixo, de onde se faz a extração, foi chamado de fonte externa de dados. Para fazer uso de um ambiente de *Data Warehousing* para armazenamento de métricas, Rêgo (2014) modelou a arquitura da solução de modo que a fonte externa de dados seriam arquivos contendo métricas de software, como pode ser visto na figura 8:

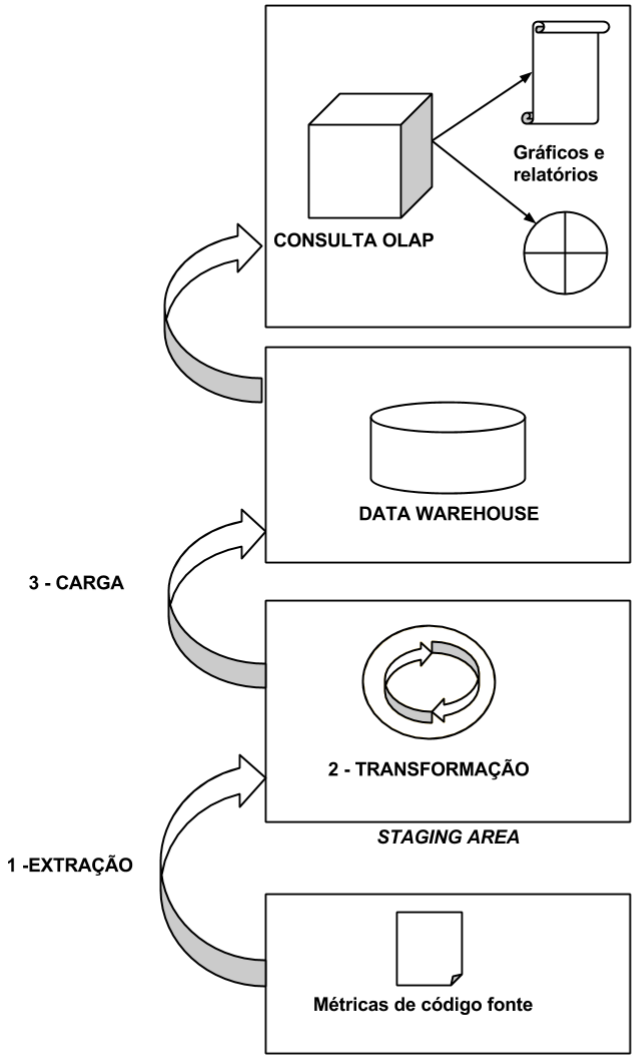

Figura 7 – Arquitetura do ambiente de *Data Warehousing* para métricas de código

Seguindo a metodologia apresentada por Kimball e Ross (2002), Rêgo (2014) definiu os seguintes requisitos de negócios 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 Open 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 o valor pecentil obtida para cada métrica de código-fonte em uma determinada release do projeto para a configuração Open JDK8 Metrics
- **Requisito 4:** Comparar o o valor pecentil a para 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

Para alcançar os requisitos definidos, Rêgo (2014) identificou os seguintes fatos e dimensões no contexto de monitoramento de métricas:

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 7 – Fatos e dimensões identificadas por Rêgo (2014)

A partir da identificação de fatos e dimensões expostos na tabela 7, Rêgo (2014) traduziu esses elementos em tabelas fato e tabelas dimensão. O resultado disso pode ser verificado na tabela 8:

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

Utilizando a ferramenta MySql Workbench, Rêgo (2014) elaborou o seguinte modelo físico baseado nos fatos e dimensões já identificados:

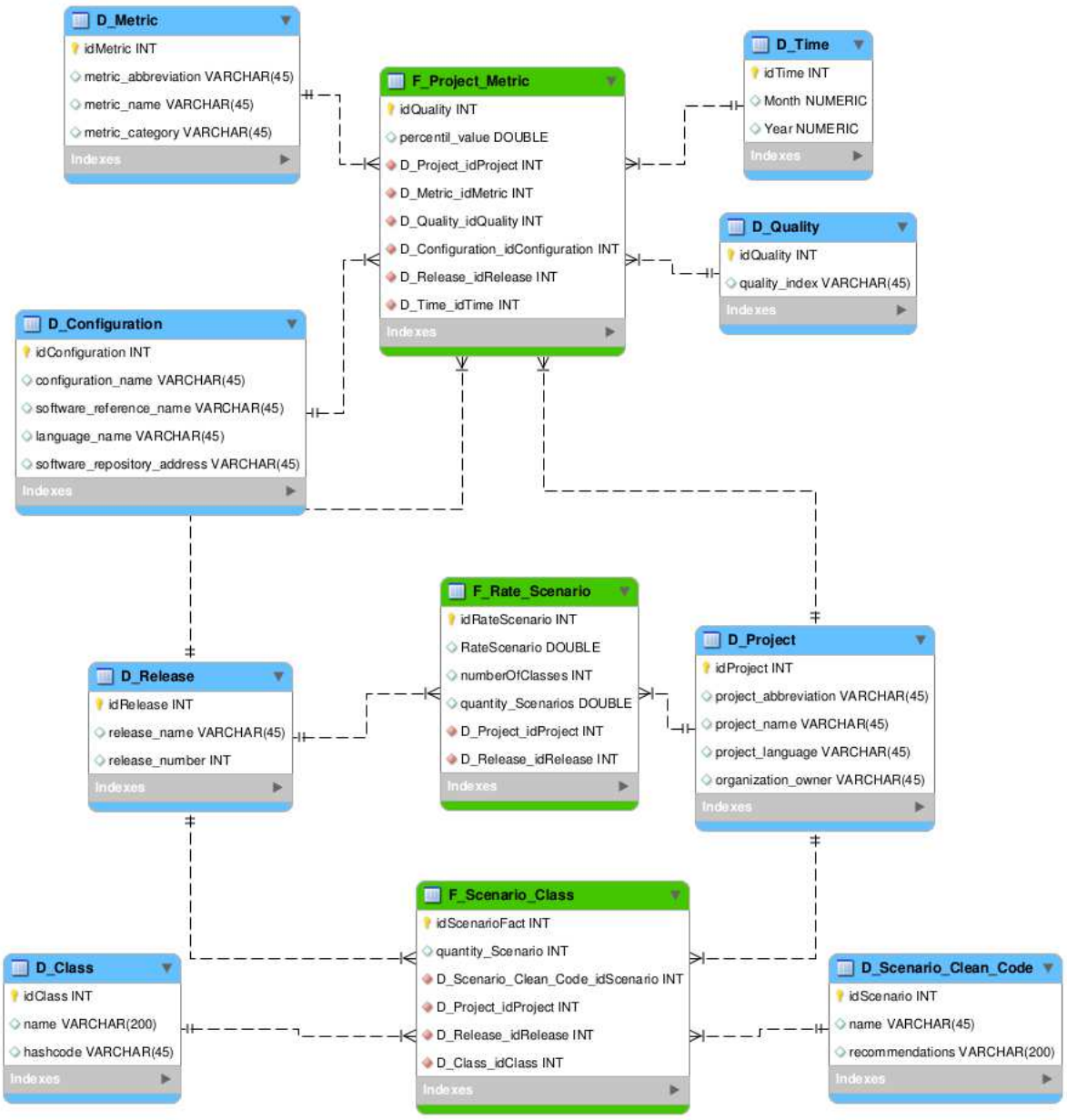

Figura 8 – Projeto físico do *Data Warehouse* extraído de Rêgo (2014)

Com o intuito de representar os dados que representam os próprios dados dos processos de negócio, Rêgo (2014) criou uma área de metadados visando facilitar o processo de *Extraction-Transformation-Load*, sendo essa uma vantagem apresentada por (KIMBALL; ROSS, 2002). A área de metadados desenvolvida possui as seguintes tabelas:

- ***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.

A figura 9 retrata o ambiente de metadados criado:

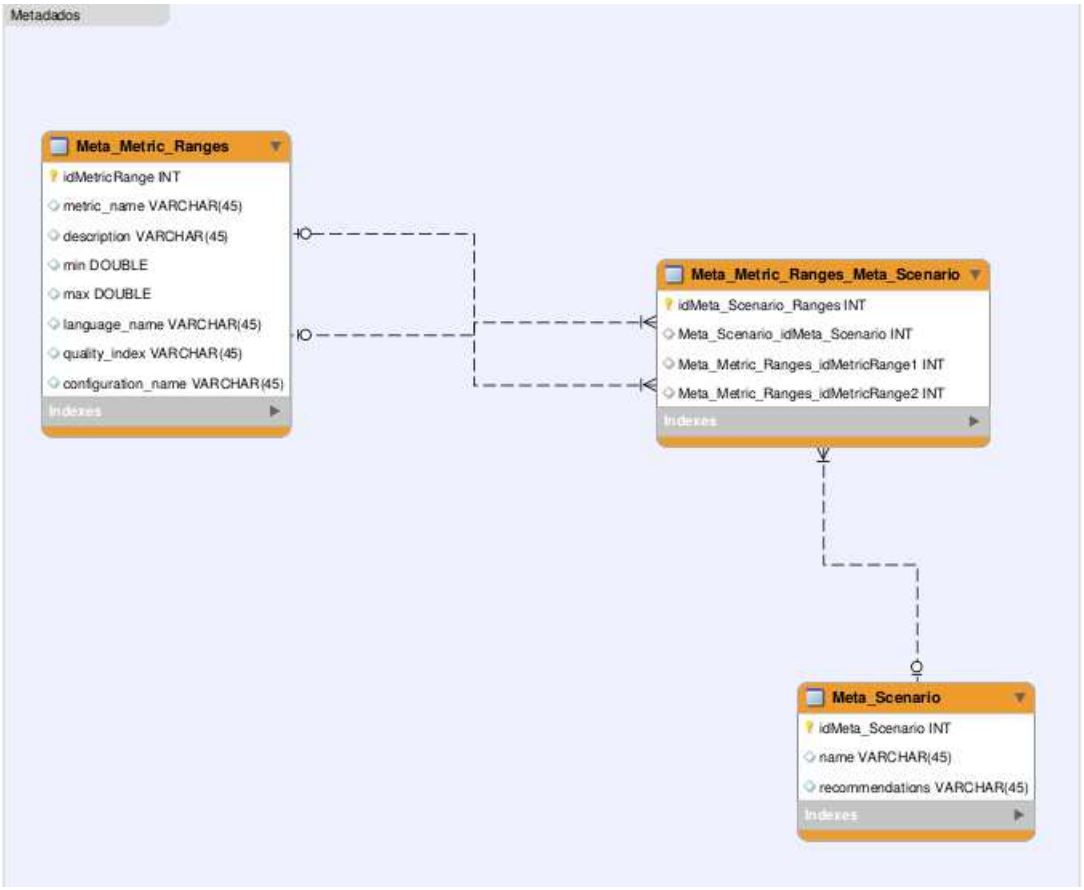

Figura 9 – Projeto físico do *Data Warehouse* extraído de Rêgo (2014)

3.5 Considerações finais do capítulo

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

4 Projeto de estudo de Caso

Este capítulo irá apresentar a estratégia de pesquisa adotada durante o trabalho, buscando elaborar um protocolo para o estudo de caso que será realizado. Elementos de pesquisa como o problema a ser resolvido e quais são os objetivos a serem alcançados serão identificados e explicados. Também será apresentado o método para coleta dos dados e como eles serão analisados.

4.1 Definição sobre estudo de caso

O estudo de caso é uma estratégia de pesquisa utilizada para investigar um tópico de manira empírica através de um conjunto de procedimentos pré-especificados (YIN, 2001). Buscando diferenciar o estudo de caso de outras estratégias de pesquisa, Yin (2001) esclarece que um estudo de caso deve focalizar acontecimentos contemporâneos, não havendo assim exigência quanto ao controle sobre os eventos comportamentais. Dessa forma, o estudo de caso difere de um experimento pelo motivo que neste há controle e manipulação sobre os eventos, diferentemente do estudo de caso, que não os manipula. Em suma, a essência de qualquer estudo de caso reside em esclarecer uma decisão ou um conjunto de decisões, considerando o motivo pelo qual elas foram tomadas e qual os resultados das suas implementações (SCHRAMM, 1971).

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

- 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 fonte dos dados coletados nessa pesquisa e qual o método de coleta?

As perguntas acima serão respondidas nas próximas seções, de modo que o estudo de caso possa ser compreendido como um projeto de pesquisa e então ser executado. Para tanto a seguinte estrutura será adotada no projeto de pesquisa:

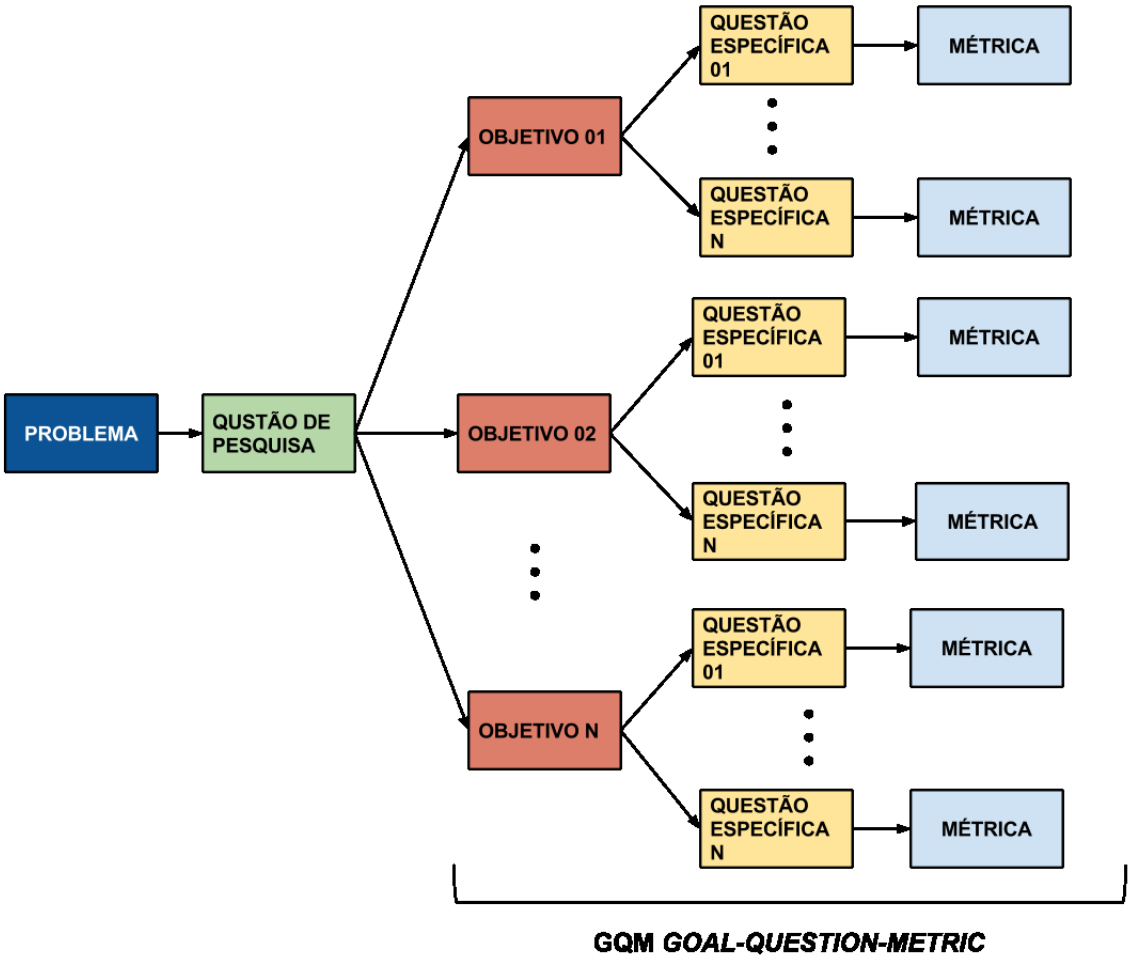

Figura 10 – Estrutura do estudo de caso

4.1.1 Problema

O PROBLEMA

4.1.2 Questão de Pesquisa

Segundo Caldiera e Rombach (1994), a questão de pesquisa deve ser capaz de caracterizar o objeto que está sendo medido, seja ele produto, processo ou recurso. Sob essa lógica, e levando em conta a análise do problema identificado, a seguinte questão de pesquisa relacionada ao processo de monitoramento de métricas de código fonte foi criada:

O uso de Data Warehousing é eficaz e eficiente no monitoramento de métricas de código fonte, facilitando a interpretação dos resultados da análise estática e apoiando decisões de refatoração?

Para atender a questão de pesquisa foi utilizado o mecanismo goal-question-metrics (GQM), usado para definir e interpretar um software operacional e mensurável. O GQM combina em si muitas das técnicas de medição e as generaliza para incorporar processos,

produtos ou recursos, o que torna seu uso adaptável a ambientes diferentes (CALDIERA; ROMBACH, 1994).

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 um 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).

Interpretação da métrica:

- Número de classes crescente e constante, Número de oportunidade de melhoria estável: Projeto cresceu mais dos que o cenários, cenários podem ou não estar sendo tratados.
- Número de classes crescente e constante, Número de oportunidade de melhoria crescendo: Projeto cresce junto com cenários que não são tratados com eficiência ou não são tratados
- Número de classes crescente ou não, mas constante, número de oportunidade de melhoria diminuindo: Projeto cresce e cenários são tratados com eficiência.

4.2 *Background*

No que se refere ao contexto desse trabalho, o principal *background* é o trabalho desenvolvido por Rêgo (2014), no qual a solução para monitoramento de métricas de

código fonte utilizando *Data Warehouse* foi desenvolvida. Novello (2006) utilizou *Data Warehouse* para monitoramento de métricas no processo de desenvolvimento de software, porém não essas métricas não eram de código fonte. Não foram encontrados outros trabalhos que usassem *Data Warehouse* para aferir qualidade do que está sendo desenvolvido relacionando métricas de código fonte a cenários de limpeza, tal qual foi feito por Rêgo (2014).

4.3 Seleção

??????????????????????????????????

4.4 Fonte dos dados coletados e método de coleta

Nesse estudo de caso, os dados foram coletados através de registros de observações em campo, questionários e resultados gerados pela própria solução após análise direta do código fonte.

Os registros oriundos de observações em campo são coletados durante encontros realizados no próprio órgão público federal com a equipe responsável pela tomada de decisões de qualidade de código fonte. Nesses encontros, a solução proposta é utilizada e qualquer atitude relacionada ao seu uso pela equipe é registrado.

A adoção de questionários foi utilizada tanto para dados qualitativos quanto para dados quantitativos. Um exemplo disso é a questão específica 06, em que se busca saber a proporção de decisões tomadas influenciadas pela solução via questionário com a empresa. O uso de questionário na questão 06 é quantitativo, enquanto na questão 08 é feito um questionário para saber o nível de satisfação da empresa quanto ao uso da solução, sendo esse um tipo de dado qualitativo.

Dados resultantes da própria ferramenta, como gráficos automaticamente gerados, serão coletados a medida que a ferramenta for utilizada ao longo do tempo.

4.5 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 do constructo, 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 do Constructo:** A validade de construção está presente na fase de coleta de dados, quando deve ser evidenciado 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). Buscou-se garantir a validade de construção ao definir objetivos com evidências diferentes. Estas, por sua vez, estão diretamente relacionadas com os objetivos do estudo de caso e os objetivos do trabalho.

- **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 deu pelo resultado da solução de *Data Warehouse* que utiliza o código-fonte (explicada no capítulo 3) , pela análise de questionários e pelos dados coletados através de entrevistas.

- **Validade externa:** Por este ser um caso único, a generalização do estudo de caso se dá de maneira pobre (YIN, 2001). Assim é necessária a utilização do estudo em múltiplos casos para que se comprove a generalidade dos resultados. Como este trabalho é o primeiro a verificar a eficácia e eficiência da solução para o estudo de caso no órgão, 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 garantem a repetibilidade desse trabalho e consequentemente a validade relacionada à confiabilidade.

4.6 Processo de análise dos dados

Análise dos dados coletados durante o estudo de caso a ser realizado no órgão público federal será realizado 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.

4.7 Considerações finais do capítulo

Esse capítulo teve como objetivo apresentar o protocolo de estudo de caso que será adotado na continuação desse trabalho. A coleta e análise dos dados coletados seguindo esse protocolo ocorrerão no próximo semestre próximo semestre.

5 Conclusão

A primeira etapa desse trabalho apresentou uma revisão bibliográfica a respeito de qualidade interna de software, sobre *Data Warehousing* e acerca da definição de estudo de caso. Os conceitos levantados na revisão bibliográfica foram utilizados durante a apresentação da solução, que utiliza *Data Warehousing* para monitoramento de métricas de código fonte. Após a revisão bibliográfica ficou clara a importância de monitorar código fonte e também quais as vantagens no uso de *Data Warehousing* para armazenamento de dados em relação a sistemas OLTP.

Após a apresentação sobre a revisão bibliográfica foi mostrado o projeto de pesquisa a ser realizado. Conceitos fundamentais a uma 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 estudo de caso.

Enquanto esta parte do trabalho teve como foco o planejamento do estudo de caso, a próxima etapa será responsável pela coleta e análise dos dados a serem obtidos, analisando a eficácia e eficiência do uso de *Data Warehousing* no monitoramento de métricas de código fonte em uma autarquia da administração pública federal, com o objetivo de responder a questão de pesquisa definida.

Na segunda parte desse trabalho será realizada também uma análise estatística da taxa de oportunidade de melhoria de código em projetos de software livre já utilizados no trabalho de (MEIRELLES, 2013). O objetivo dessa análise é obter um intervalo qualitativo para essa taxa de oportunidade, possibilitando assim que a eficácia da solução proposta possa ser aferida com mais facilidade e exatidão no estudo de caso. Além disso, serão criados mais cenários de limpeza relacionando-os a um conjunto de métricas de código.

Referências

ALMEIDA, L. T.; MIRANDA, J. M. de. Código limpo e seu mapeamento para métricas de código fonte. 2010. Disponível em: <http://ccsl.ime.usp.br/pt-br/system/files/relatorio-codigo-limpo.pdf>. Citado 4 vezes nas páginas 10, 26, 27 e 28.

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 22.

BECK, K. *Implementation Patterns*. 1. ed. [S.l.]: Addison-Wesley Professional, 2007. Citado na página 26.

CALDIERA, V.; ROMBACH, H. D. The goal question metric approach. v. 2, n. 1994, p. 528–532, 1994. Disponível em: <http://www.csri.utoronto.ca/~sme/CSC444F/handouts/GQM-paper.pdf>. Citado 2 vezes nas páginas 44 e 45.

CHAUDHURI, S.; DAYAL, U. An overview of data warehousing and olap technology. *ACM Sigmod record*, ACM, v. 26, n. 1, p. 65–74, 1997. Citado na página 31.

FENTON, N. E.; PFLEEGER, S. L. *Software Metrics: A Rigorous and Practical Approach*. 2 edition. ed. [S.l.]: Course Technology, 1998. 656 p. Citado 3 vezes nas páginas 15, 19 e 20.

GILMORE, W. J. *Dominando PHP e MySql*. [S.l.]: STARLIN ALTA CONSULT, 2008. Citado na página 22.

INMON, W. H. *Building the Data Warehouse*. 3rd. ed. New York, NY, USA: John Wiley & Sons, Inc., 2002. Citado na página 31.

ISO/IEC 15939. *ISO/IEC 15939: Software Engineering - Software Measurement Process*. [S.l.], 2002. Citado na página 19.

ISO/IEC 25023. *ISO/IEC 25023: Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - Measurement of system and software product quality*. [S.l.], 2011. Citado 3 vezes nas páginas 9, 15 e 21.

ISO/IEC 9126. *ISO/IEC 9126-1: Software Engineering - Product Quality*. [S.l.], 2001. Citado 2 vezes nas páginas 19 e 20.

KAN, S. H. *Metrics and models in software quality engineering*. [S.l.]: Addison Wesley, 2002. Citado 3 vezes nas páginas 21, 22 e 23.

KIMBALL, R. *The data warehouse lifecycle toolkit: expert methods for designing, developing, and deploying data warehouses*. [S.l.]: Wiley. com, 1998. Citado na página 35.

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 6 vezes nas páginas 32, 33, 34, 35, 37 e 41.

LAIRD, M. C. B. L. M. *Software measurement and estimation: A practical approach*. [S.l.]: Wiley-IEEE Computer Society Press, 2006. Citado na página 22.

MCCABE, T. J. A Complexity Measure. *IEEE Transactions Software Engineering*, v. 2, n. 4, p. 308–320, December 1976. Citado na página 21.

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 7 vezes nas páginas 10, 19, 21, 22, 23, 24 e 50.

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 4 vezes nas páginas 10, 34, 35 e 36.

NOVELLO, T. C. *Uma abordagem de Data Warehouse para análise de processos de desenvolvimento de software*. phdthesis, 2006. Disponível em: <http://tardis.pucrs.br/dspace/handle/10923/1570>. Citado na página 47.

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 21 vezes nas páginas 9, 10, 15, 16, 21, 22, 24, 25, 26, 27, 28, 29, 36, 37, 38, 39, 40, 41, 42, 46 e 47.

SCHRAMM, W. Notes on case studies of instructional media projects. 1971. Disponível em: <http://eric.ed.gov/?id=ED092145>. Citado na página 43.

SHARMA, N. *Getting started with data warehousing*. [S.l.]: IBM Redbooks, 2011. Citado 5 vezes nas páginas 9, 31, 32, 35 e 36.

TIMES, V. C. *Sistemas de DW*. 2012. Universidade Federal de Pernambuco - UFPE. Disponível em: <www.cin.ufpe.br/~if695/bda_dw.pdf>. Citado 3 vezes nas páginas 9, 34 e 35.

WOHLIN, C. et al. *Experimentation in software engineering*. [S.l.]: Springer, 2012. Citado na página 16.

YIN, R. *Estudo de caso: planejamento e métodos*. [S.l.]: Bookman, 2001. Citado 3 vezes nas páginas 43, 47 e 48.