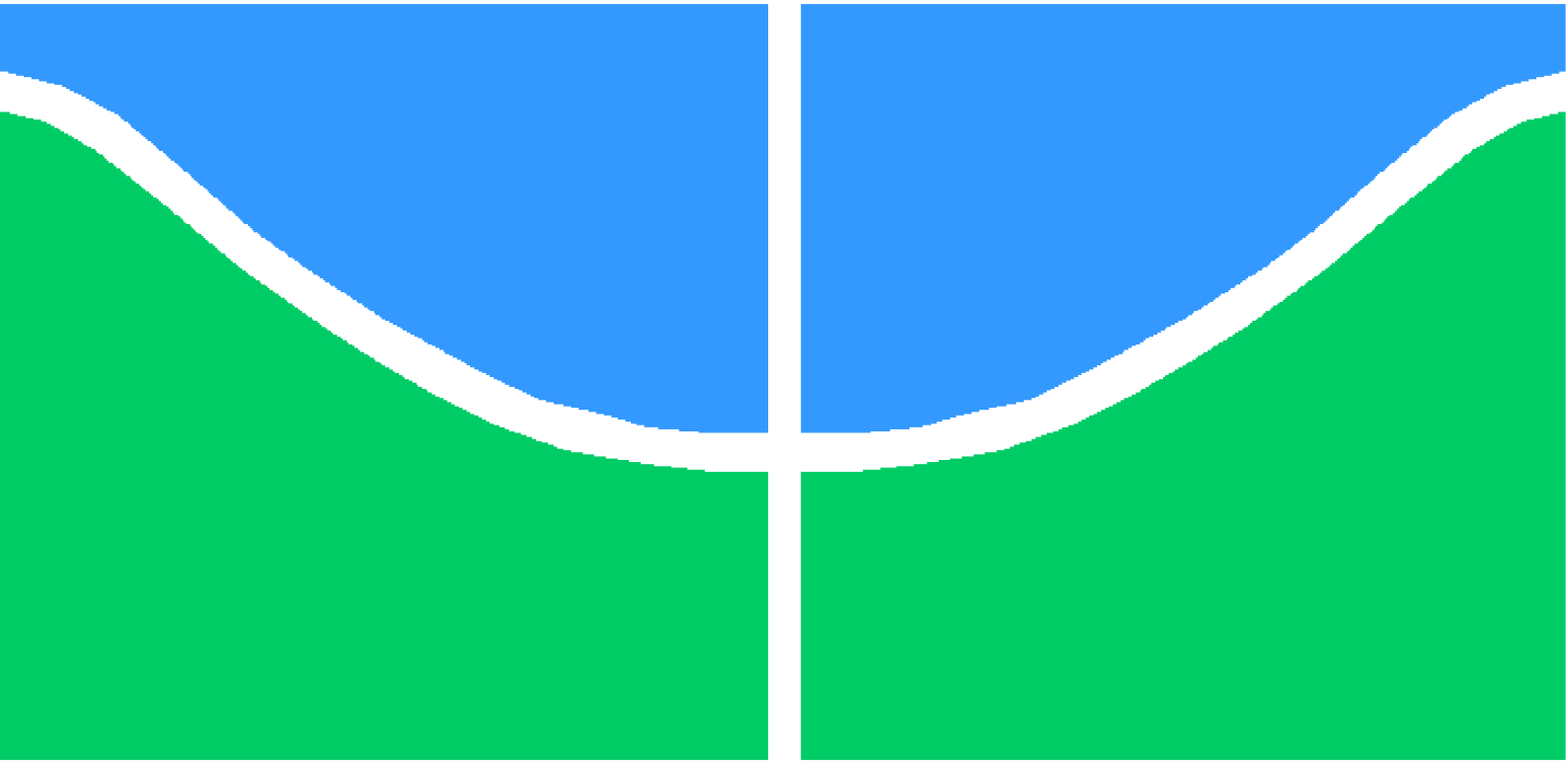

Universidade de Brasília - UnB

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 no Tribunal de Contas da União

Autores: Matheus Oliveira Tristão dos Anjos

Pedro da Cunha Tomioka

Orientador: Prof. Msc. Hilmer Rodrigues Neri

Brasília, DF

2014

Matheus Oliveira Tristão dos Anjos
Pedro da Cunha Tomioka

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 no Tribunal de Contas da União

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
Pedro da Cunha Tomioka

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 no Tribunal de Contas da União/ Matheus Oliveira Tristão dos Anjos & Pedro da Cunha Tomioka. – Brasília, DF, 2014-

58 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 no Tribunal de Contas da União

CDU 02:141:005.6

Matheus Oliveira Tristão dos Anjos
Pedro da Cunha Tomioka

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 no Tribunal de Contas da União

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, 18 de Novembro de 2014:

Prof. Msc. Hilmer Rodrigues Neri
Orientador

A confirmar
Convidado 1

A confirmar
Convidado 2

Brasília, DF
2014

Agradecimentos

de Matheus Oliveira Tristão dos Anjos

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.

Agradecimentos

de Pedro da Cunha Tomioka

A Universidade de Brasília pela oportunidade fazer o cruseo.

Ao professor Hilmer Rodrigues Neri pela oportunidade e apoio na elaboração deste trabalho, e também pelo ensinos e conhecimento fornecidos dentro e fora da sala de aula.

Aos meus pais pelo incentivo e preocupação por em todos estes anos.

Aos meus amigos de graduação Matheus Tristão, Nilton César Araruna e Guilherme Baufaker, pelos momentos de dedicação ao estudo e pela amizade cultivada durante o tempo do curso.

Ao meu primeiro chefe Odnalro Cruz Videira Júnior, por acreditado em mim, por ter sempre me incentivado a melhorar e por me mostrar o caminho que devo seguir para ser um bom profissional.

Por fim agradeço a meus amigos e companheiros de graduação: Pedro Henrique Potiguara, Aline Gonçalves, Hichemm Khalyd, Guilherme de Lima, Tiago Pereira, Guilherme Fay, Pedro Matias, Mayco Henrique, Carlos Filipe, Marcos Ronaldo, Rafael Ferreira, Matheus Braga, Thabata Granja, Fágner Rodrigues, Bruno Motta, Paulo Acés, Matheus Patrocínio, Greg Ouyama, Eusyar Alves, João Pedro Carvalho, Artur Potiguara, Pedro Guilherme Moreira e Pedro Inazawa.

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

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 TCU.

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 the TCU.

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

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*
DIT	*Depth of Inheritance Tree*
DW	*Data Warehouse*
DSS	*Secision-Support Systems*
ETL	*Extraction-Transformation-Load*
GQM	*Goal-Question-Metric*
IEC	*International Electrotechnical Commission*
ISO	*International Organization for Standardization*
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*
TCU	Tribunal de Contas da União

Sumário

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).

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 afetam a qualidade do produto como um todo. Do ponto de vista da engenharia de software, métricas de código fonte de um software fornecem uma maneira quantitativa de avaliar a qualidade de atributos internos do produto, habilitando assim a avaliação da qualidade antes do produto ser concluído (PRESSMAN, 2010).

Além disso, o processo de medição da qualidade interna do software possibilita também 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).

1.2 Problema

Apesar de métricas de código-fonte serem coletadas facilmente com o auxílio de ferramentas de análise estática de código, como, por exemplo, por meio do *Sonar* e o *Analizo*, sua interpretação ainda pode ser um desafio que se não for superado pode tornar essa coleta algo sem muito valor observável. Ferramentas de análise de métricas frequentemente exportam seu resultado como valores numéricos isolados (MEIRELLES, 2013) e, 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 que melhore sua qualidade.

A dificuldade em interpretar os dados disponibilizados por tais ferramentas de análise do código afeta a avaliação de projetos de software, onde são um fator essencial para a tomada de decisão à nível de gerenciamento de projeto, de processo, de engenharia de software e de outros processos de apoio (PANDIAN, 2004). Surge a partir desse fato a necessidade de uma solução que apoie a tomada de decisão, associando esses valores numéricos à uma semântica de interpretação como cenários de limpeza. Cenários de limpeza

fazem parte de um conceito levantado por Machini et al. (2010), em que foi realizado mapeamento entre as métricas de código-fonte e as técnicas e práticas propostas por Martin (2008) e Beck (2007).

Buscando facilitar a interpretação das métricas de código fonte, bem como apoiar as decisões de refatoração a serem tomadas, Rêgo (2014) construiu uma solução que faz uso de um ambiente de DW para armazenamento e monitoramento de métricas de código-fonte. Dá-se então a necessidade de investigar, por meio de estudo de caso, a eficácia e eficiência da solução de DW proposta no contexto do problema descrito. Assim foi criada a seguinte questão geral de pesquisa:

O uso de um ambiente de Data Warehousing para aferição e monitoramento da qualidade interna do código-fonte é eficaz e eficiente do ponto de vista da equipe de qualidade?

1.3 Objetivos

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

- levantar fundamentação teórica que servirá de base para todo o trabalho;
- analisar solução desenvolvida por Rêgo (2014) para monitoramento de métricas por meio de um ambiente de *Data Warehousing*;
- definir, projetar e caracterizar o do estudo de caso.

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

- realizar um estudo com uso de estatística descritiva para analisar o comportamento da taxa de oportunidade de melhoria de código-fonte, apresentada na solução. Espera-se identificar com esse estudo, padrões, tendências e possivelmente, valores de referência para interpretação desse indicador;
- 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;
- coletar dados de um projeto em desenvolvimento na organização selecionada;
- realizar análise dos dados coletados;

- relatar resultados obtidos.

1.4 Metodologia de Pesquisa

Nessa seção será definida a metodologia da pesquisa, definindo qual a natureza da pesquisa, a forma de abordagem, o tipo de objetivo que se espera alcançar, os procedimentos técnicos e o tipo de coleta de dados. A figura 1 apresenta um esquema para a metodologia, buscando classificar a pesquisa quanto aos elementos que a classificam segundo Silva e Menezes (2005):

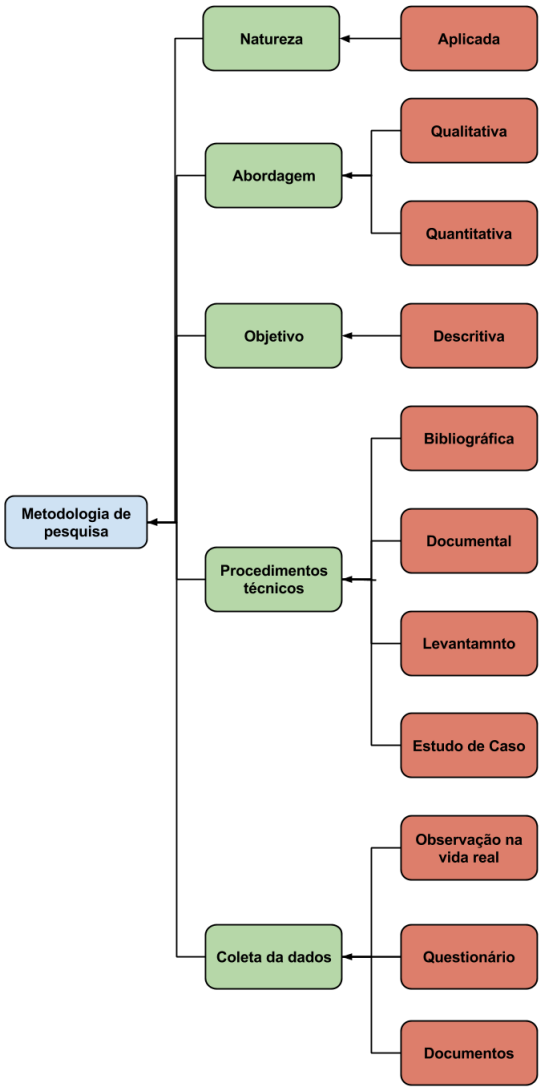

Figura 1 – Metodologia de pesquisa

Do ponto de vista da natureza, uma pesquisa pode ser considerada básica ou aplicada (SILVA; MENEZES, 2005). A pesquisa a qual esse trabalho se refere é aplicada

pois seus resultados contém aplicação prática e são dirigidos à solução de problemas específicos.

A abordagem da pesquisa será tanto qualitativa quanto quantitativa, pois ao mesmo tempo em que busca classificar informações em números através de técnicas estatísticas, também analisa a subjetividade de algumas questões sem que isso seja traduzido em números.

O objetivo que essa pesquisa busca alcançar faz com que ela seja considerada descritiva, pois uma pesquisa descritiva visa descrever as características de determinado fenômeno envolvendo uso de técnicas padronizadas de coleta de dados (SILVA; MENEZES, 2005). Quanto aos procedimentos técnicos ela pode ser considerada de levantamento, pois envolve a interrogação direta das pessoas cujo comportamento se deseja conhecer. Além disso, do ponto de vista dos procedimentos técnicos, ela também pode ser considerada bibliográfica, documental e como um estudo de caso.

Também foi modelado no esquema da metodologia quais serão as formas de coleta de dados. Como pode pode ser visto na figura 1, a coleta é feita através de questionários, obervação na vida real e documentos, que incluem o próprio relatório da solução adotada no estudo de caso.

As etapas necessárias para o desenvolvimento do projeto de estudo de caso se baseiam na diagramação de pesquisa de Gil (2002), nos passos para elaborar um estudo de caso elicitados por Yin (2001) e no protocolo de estudo de caso de Brereton, Kitchenham e Budgen (2008). A figura 2 apresenta os elementos do projeto com sua ordem de execução em forma de fluxo.

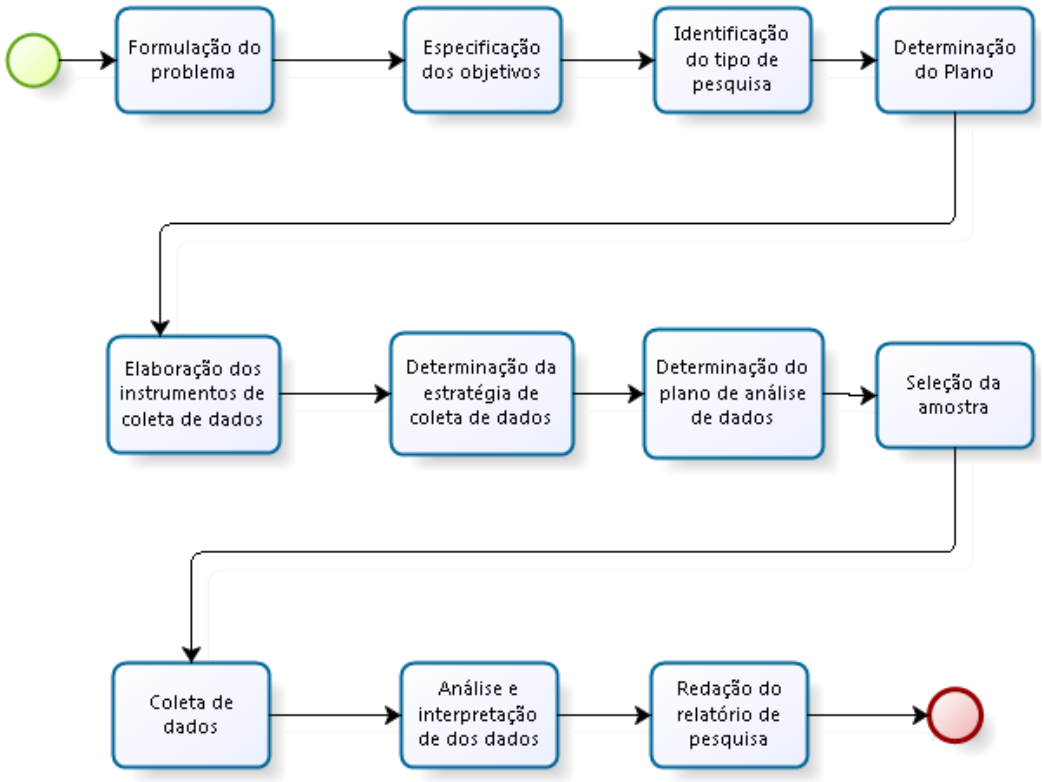

Figura 2 – Diagramação do projeto de estudo de caso

Os passos específicos do estudo de caso são apresentados após a identificação do tipo de pesquisa. Assim a partir do passo de determinação do plano até o passo de seleção da amostra são apresentados o projeto de estudo de caso onde serão identificados elementos como o problema a ser resolvido, a questão de pesquisa e demais tópicos definidos no protocolo de estudo de caso. Em seguida, a coleta de dados é feita através de questionários, entrevistas informais, observações em campo e resultados provenientes da solução de DWing que será utilizada. A etapa de análise e interpretação dos dados refere-se às atividades de categorização, exibição, verificação e conclusão de dados de natureza qualitativa e quantitava. Por fim, a redação do relatório de pesquisa apresenta a redação dos resultados de forma adequada para o leitor alvo.

1.5 Organização do Trabalho

Este 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:** São empregadas expressões semânticas para representar objetos para fins de identificação (PANDIAN, 2004). Na interpretação dos valores de cada atributo, a ordem não possui significado (MEIRELLES, 2013).

- **Ordinal:** Os valores podem ser comparados em ordem, é possível agrupar valores em categorias também podem ser ordenadas. Porém esta escala não oferece informações sobre a magnitude da diferença entre os elementos (KAN, 2002).

- **Intervalo:** A ordem dos resultados é importante, assim como o tamanho dos intervalos que separam os pontos, mas a proporções não são necessariamente válidas (MEIRELLES, 2013). Operações matemáticas como adição e subtração podem ser aplicadas a este tipo de intervalo (KAN, 2002).

- **Racional:** Possui a mesma definição da escala de intervalo, a diferença está no fato da proporção ser preservada (MEIRELLES, 2013). Ou seja, é possível definir um unidade zero para um tamanho de unidade previamente estabelecido pela escala (KAN, 2002).

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 3 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:

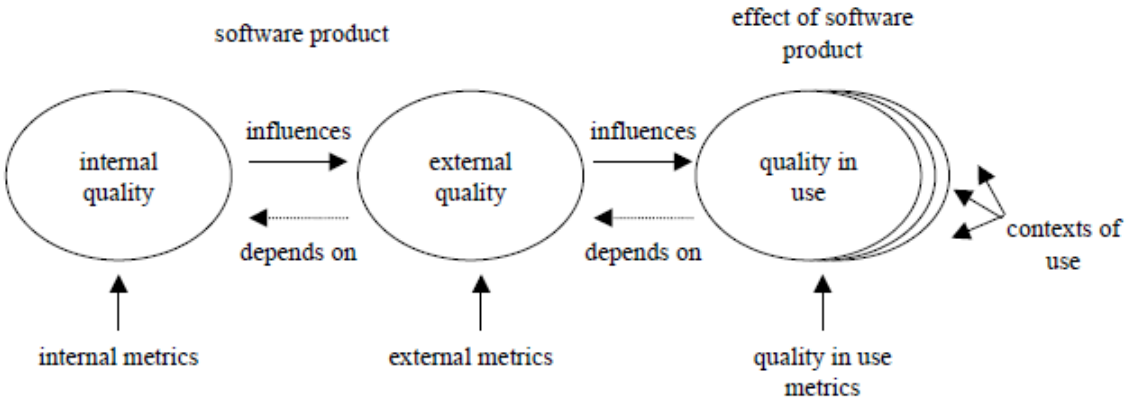

Figura 3 – Modelo de Qualidade do Produto adaptado da ISO/IEC 25023 (2011)

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). Segundo Pressman (2010) é provavél que a reusabilidade de uma classe diminua à medida que o CBO aumenta, pois valores altos complicam as modificações e os testes.

- **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). Quanto mais profunda for a árvore, maior será o número de classes envolvidas (CHIDAMBER; KEMERER, 1994), consequentemente haverá um número maior de métodos e atributos herdados, aumentando assim a complexidade da classe Meirelles (2013).

- **LCOM4** (*Lack of Cohesion in Methods* - Falta de Coesão entre Métodos):Falta de Coesão entre Métodos): Proposta inicialmente por Chidamber e Kemerer (1994), LCOM calcula o número de métodos que têm acesso a um ou mais atributos de uma dada classe (PRESSMAN, 2010). Por ter recebido diversas críticas, várias alternativas foram criadas. Hitz e Montazeri (1995) definiu uma revisão desta métrica e assim elaborou uma outra versão conhecida como LCOM4. Para calcular LCOM4 de um módulo, é necessário construir um gráfico não-orientado em que os nós são os métodos e atribuos de uma classe. Para cada método, deve haver uma areste entre

ele e um outro método ou variável que ele usa. O valor da LCOM4 é o número de componentes fracamente conectados nesse 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. Pressman (2010) justifica esse fato afirmando que à medida que o número de filhos cresce, alguns deles não são necessariamente membros adequados da super classe.

- **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 Bom Regular Ruim	[de 0 a 33] [de 34 a 87] [de 88 a 200] [acima de 200]	[de 0 a 33] [de 34 a 105] [de 106 a 276] [acima de 276]
ACCM	Excelente Bom Regular Ruim	[de 0 a 2,8] [de 2,9 a 4,4] [de 4,5 a 6,0] [acima de 6]	[de 0 a 3] [de 3,1 a 4,0] [de 4,1 a 6,0] [acima de 6]
AMLOC	Excelente Bom Regular Ruim	[de 0 a 8,3] [de 8,4 a 18] [de 19 a 34] [acima de 34]	[de 0 a 8] [de 8,1 a 16,0] [de 16,1 a 27] [acima de 27]
ACC	Excelente Bom Regular Ruim	[de 0 a 1] [de 1,1 a 5] [de 5,1 a 12] [acima de 12]	[de 0 a 1,0] [de 1,1 a 5,0] [de 5,1 a 13] [acima de 13]
ANPM	Excelente Bom Regular Ruim	[de 0 a 1,5] [de 1,6 a 2,3] [de 2,4 a 3,0] [acima de 3]	[de 0 a 2,0] [de 2,1 a 3,0] [de 3,1 a 5,0] [acima de 5]
CBO	Excelente Bom Regular Ruim	[de 0 a 3] [de 4 a 6] [de 7 a 9] [acima de 9]	[de 0 a 2] [de 3 a 5] [de 5 a 7] [acima de 7]
DIT	Excelente Bom Regular Ruim	[de 0 a 2] [de 3 a 4] [de 5 a 6] [acima de 6]	[de 0 a 1] [de 2 a 3] [de 3 a 4] [acima de 4]
LCOM4	Excelente Bom Regular Ruim	[de 0 a 3] [de 4 a 7] [de 8 a 12] [acima de 12]	[de 0 a 3] [de 4 a 7] [de 8 a 11] [acima de 11]
NOC	Excelente Bom Regular Ruim	[0] [1 a 2] [3] [acima de 3]	[1] [1 a 2] [3] [acima de 3]
NOM	Excelente Bom Regular Ruim	[de 0 a 8] [de 9 a 17] [de 18 a 27] [acima de 27]	[de 0 a 10] [de 11 a 21] [de 22 a 35] [acima de 35]
NPA	Excelente Bom Regular Ruim	[0] [1] [de 2 a 3] [acima de 3]	[0] [1] [de 2 a 3] [acima de 3]
RFC	Excelente Bom Regular Ruim	[de 0 a 9] [de 10 a 26] [de 27 a 59] [acima de 59]	[de 0 a 11] [de 12 a 30] [de 31 a 74] [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, Machini et al. (2010) levantou 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 Machini et al. (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 Machini et al. (2010) extraídos de Rêgo (2014)

Após o levantamento dos conceitos de limpeza, Machini et al. (2010) criou 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 Machini et al. (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 Definições e terminologia

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.

São vários os tipos de aplicação que um *data warehouse* suporta. Algumas delas serão definidas a seguir.

OLAP (*online analytical processing* – processamento analítico on-line): Conjunto de princípios que fornecem uma estrutura dimensional de apoio à decisão (KIMBALL; ROSS, 2002). As ferramentas OLAP empregam as capacidades de computação distribuída para análises que requerem mais armazenamento e poder de processamento do que pode estar localizado econômica e eficientemente em um *desktop* individual (EL-

MASRI; NAVATHE, 2011). Nestas aplicações dados sumariados e históricos são mais importantes que dados atômicos (NERI, 2002).

DSS (*decision-support systems*– sistemas de apoio à decisão): são sistemas que ajudam os principais tomadores de decisões de uma organização com dados de nível mais alto em decisões complexas e importantes (ELMASRI; NAVATHE, 2011). Segundo Kimball (2008) os DSSs têm como objetivo tornar a informação de uma organização acessível e consistente, prover uma fonte adaptável e resiliente de informações, e garantir a segurança aos dados para assim ser um sistema base para a tomada de decisão.

OLTP (*online transaction processing* – processamento on-line de transações): bancos de dados tradicionais têm suporte para o OLTP, com suas operações de inserção, atualização e exclusão e também à consulta de dados. Sharma (2011) explica que sistemas OLTP usam tabelas simples para armazenar dados, estes que são normalizados, para se reduzir a redundância ou até mesmo eliminá-los, buscando sempre a garantia de sua consistência.

3.2 Características dos *Data Warehouses*

Elmasri e Navathe (2011) diferencia um *data warehouse* de uma estratégia de multibancos de dados afirmando que o primeiro possui um armazenamento em um modelo multidimensional, assim dados de múltiplas fontes são integrados e processados para tal e sendo assim contrário ao segundo, que oferece acesso a bancos de dados disjuntos e normalmente heterogêneos. A vantagem se nota pela adoção de um padrão de design mais conciso, que pode levar a uma menor complexidade de implantação.

Diferentemente da maioria dos banco de dados transacionais, *data warehouses* costumam apoiar a análise de série temporal e tendência, ambas exigindo mais dados históricos do que geralmente é mantido nos bancos de dados transacionais. É importante notar também que os *data warehouses* são não-voláteis. Sendo assim, suas informações possuem a característica de mudarem de uma forma muito menos frequente, portanto dificilmente seria enquadrada como uma informação em tempo real, e sim como uma de atualização periódica (ELMASRI; NAVATHE, 2011).

Elmasri e Navathe (2011) ordena as seguintes características diferenciadoras de data warehouses, proveniente da lista original de Codd, Codd e Salley (1993) sobre OLAP:

- Visão conceitual multidimensional.
- Dimensionalidade genérica.
- Dimensões e níveis de agregação ilimitados.
- Operações irrestritas entre dimensões.

- Tratamento dinâmico de matriz esparsa.
- Arquitetura de cliente-servidor.
- Suporte para múltiplos usuários.
- Acessibilidade.
- Transparência.
- Manipulação de dados intuitiva.
- Desempenho de relatório consistente.
- Recurso de relatório flexível.

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 principais diferenças entre aplicações OLTP e OLAP extraídas do trabalho de Neri (2002):

OLTP	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)

3.2.1 Componentes Básicos de um Data Warehouse

A Figura 4 representa a estrutura básica de um *data warehouse* segundo Kimball e Ross (2002). Os componentes apresentados serão definidos em seguida.

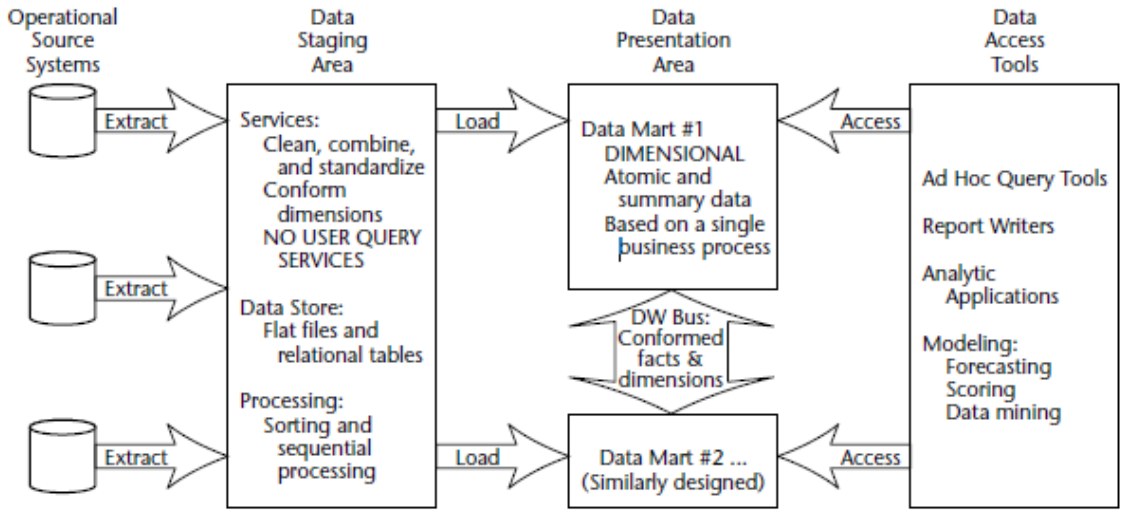

Figura 4 – Elementos básicos de um *data warehouse* extraído de Kimball e Ross (2002)

Sistemas de Fonte de Dados Operacionais (OSS - *Operational Source Systems*): Representa os sistemas que irão prover informações para o *data warehouse*. Seus dados podem ser provenientes de outros sistemas OLTPs, planilhas e etc, que compõem o negócio a ser tratado. Devem ser pensados como fora do *data warehouse* porque há pouco ou nenhum controle sobre o conteúdo e formato dos dados presentes nestes tipos de sistemas legados (KIMBALL; ROSS, 2002).

Área de Preparação dos Dados (DSA - *Data Staging Area*): É onde irá ocorrer a possível limpeza e reformatação dos dados antes que sejam carregados no *data warehouse* (ELMASRI; NAVATHE, 2011). Essas tarefas caracterizam os passos do processo que consistem 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).

Área de Apresentação dos Dados (DPA - *Data Presentation Area*):

Representa a área onde os dados são organizados, armazenados e disponibilizados para consulta direta pelos usuários, autores de relatórios e outras aplicações. Os dados da área de apresentação devem ser dimensionais e atômicos e devem estar de acordo com a arquitetura do *data warehouse* (KIMBALL; ROSS, 2002).

Ferramentas de Acesso de Dados (DAT - *Data Access Tools*): Representa a área onde uma variedade de ferramentas e técnicas podem ser utilizadas para apresentar aos usuários de negócio as consultas feitas aos dados do *data warehouse*. A apresentação dos dados nessa área deve prover insumos que irão ajudar na tomada de decisões analíticas (KIMBALL; ROSS, 2002)

Metadados: Definido como toda a informação no ambiente de *data warehouse* que não são os dados em si (KIMBALL; ROSS, 2002). Os metadados num ambiente *data warehouse* podem apontar para dados sobre tabelas do sistema, índices, relacionamentos, e etc. Kimball (1998) recomenda que a arquitetura de um DW seja orientada a metadados, devido a seu papel crítico de prover informações e parâmetros que permitem que as aplicações executarem suas tarefas com um controle maior sobre os dados provenientes das fontes de dados e outros elementos fundamentais para sua execução.

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.

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:

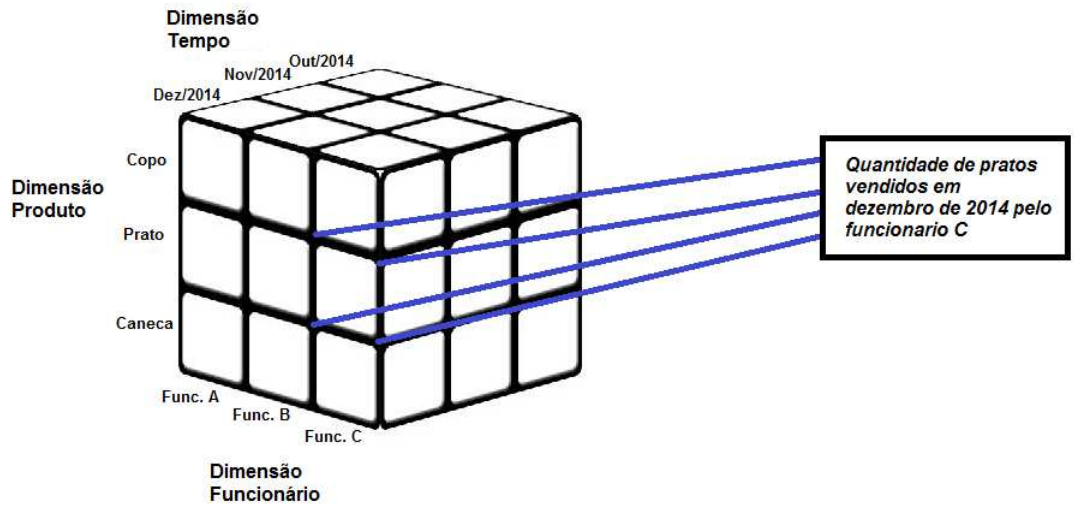

Figura 5 – Cubo de dados com o fato Venda e dimensões Produto, Funcionário e Tempo

As operações possíveis de serem aplicadas a um cubo OLAP são categorizadas a seguir.

- ***Drill Down:*** *Drilling* em modelagem multidimensional significa ir de um nível hierárquico a outro (BALLARD, 2006). Portanto, *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:

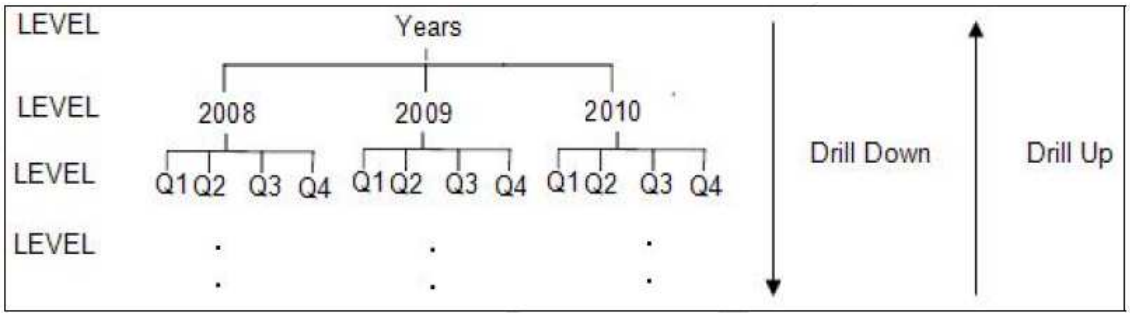

Figura 6 – Diferença entre o fluxo seguido pelo Drill Up e pelo Drill Down, extraída de (SHARMA, 2011)

3.3.1 Tipos de Tabelas do Modelo Multidimensional

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.

3.3.2 Esquemas Multidimensionais

Dois esquemas comuns são o esquema estrela e o esquema floco de neve. O esquema estrela consiste em uma tabela de fatos com uma única tabela para cada dimensão, como pode ser visto na Figura 7 (ELMASRI; NAVATHE, 2011). O esquema floco de neve é resultado da normalização e expansão das tabelas de dimensão do esquema estrela (BALLARD, 2006). Segundo Kimball e Ross (2002) o floco de neve não é recomendado em uma ambiente de *data warehouse*, pois quase sempre faz com que a apresentação dos dados ao usuário seja mais complexa, além do impacto negativo que causa sobre o desempenho da navegação.

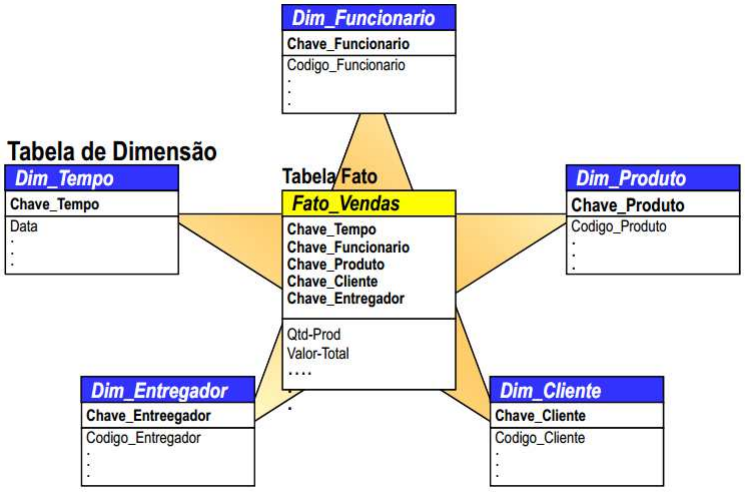

Figura 7 – Esquema estrela extraído de Times (2012)

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

A Figura 4 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 arquitetura da solução de modo que a fonte externa de dados seriam arquivos contendo métricas de software (Figura 8).

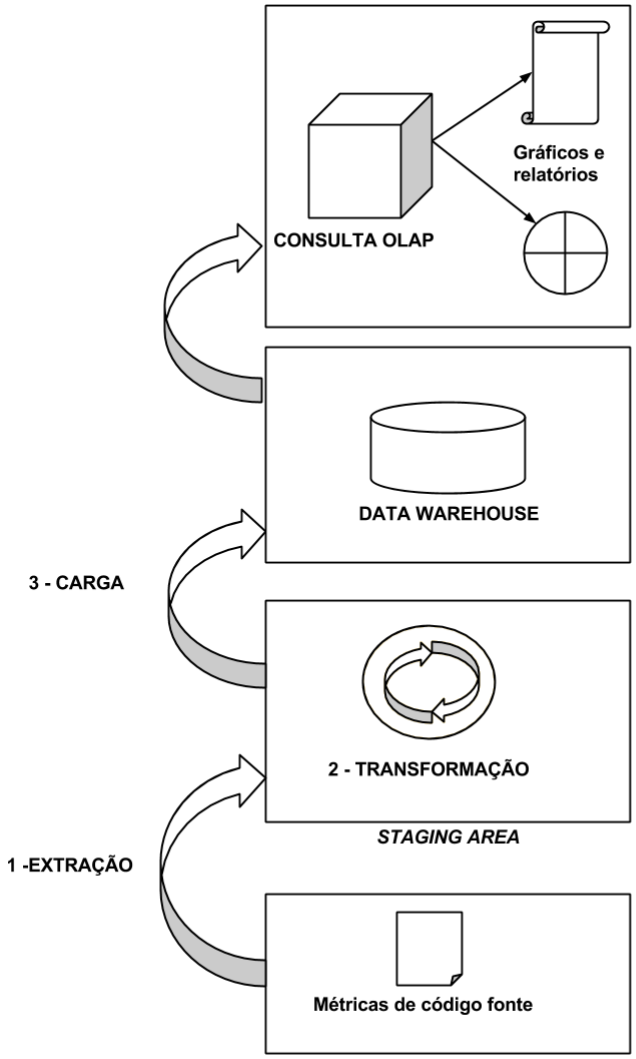

Figura 8 – Arquitetura do ambiente de *Data Warehousing* para métricas de código

Seguindo a metodologia de Kimball e Ross (2002) para projetar um *data warehouse*, o projeto de Rêgo (2014) para a solução de *data warehousing* para métricas de código-fonte seguiu os passos da metodologia, definidos em cada sub-seção a seguir.

3.4.1 Seleção do Processo de Negócio

Rêgo (2014) buscou identificar os processos de negócio e seus respectivos requisitos. Assim foram levantados 15 requisitos de negócio, sendo que os oito primeiros se referem a avaliação dos valores percentis das métricas de código-fonte e o restante a avaliação de cenários de limpeza e taxa de aproveitamento de oportunidades de melhoria de código-fonte. Os requisitos levantados por Rêgo (2014) foram os seguintes:

- **Requisito 1:** Visualizar o intervalo qualitativo obtido para cada métrica de código-fonte em uma determinada release do projeto para a configuração Open JDK8 Me-

trics.

- **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 percentil 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 percentil 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.

3.4.2 Verificação da Periodicidade de Coleta de Dados

A identificação da periodicidade de coleta dos dados é essencial para que esta seja realizada de maneira correta, além de viabilizar a agregação dos dados em níveis ou hierarquias (RêGO, 2014). Assim a periodicidade foi identificada como as *releases* do software.

3.4.3 Identificação dos Fatos e das Dimensões

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:

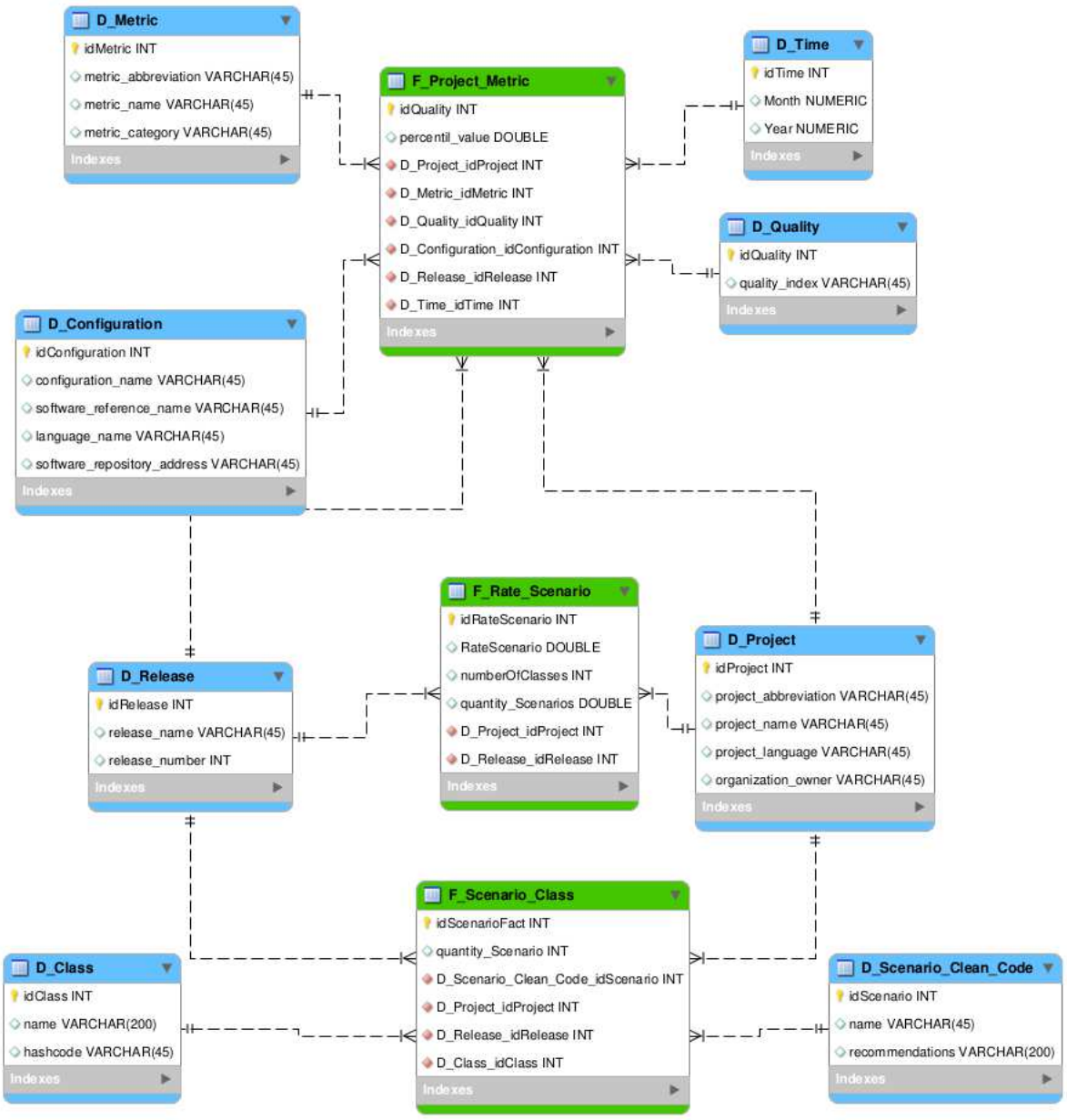

Figura 9 – Projeto físico do *Data Warehouse* extraído de Rêgo (2014)

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 10 retrata o ambiente de metadados criado:

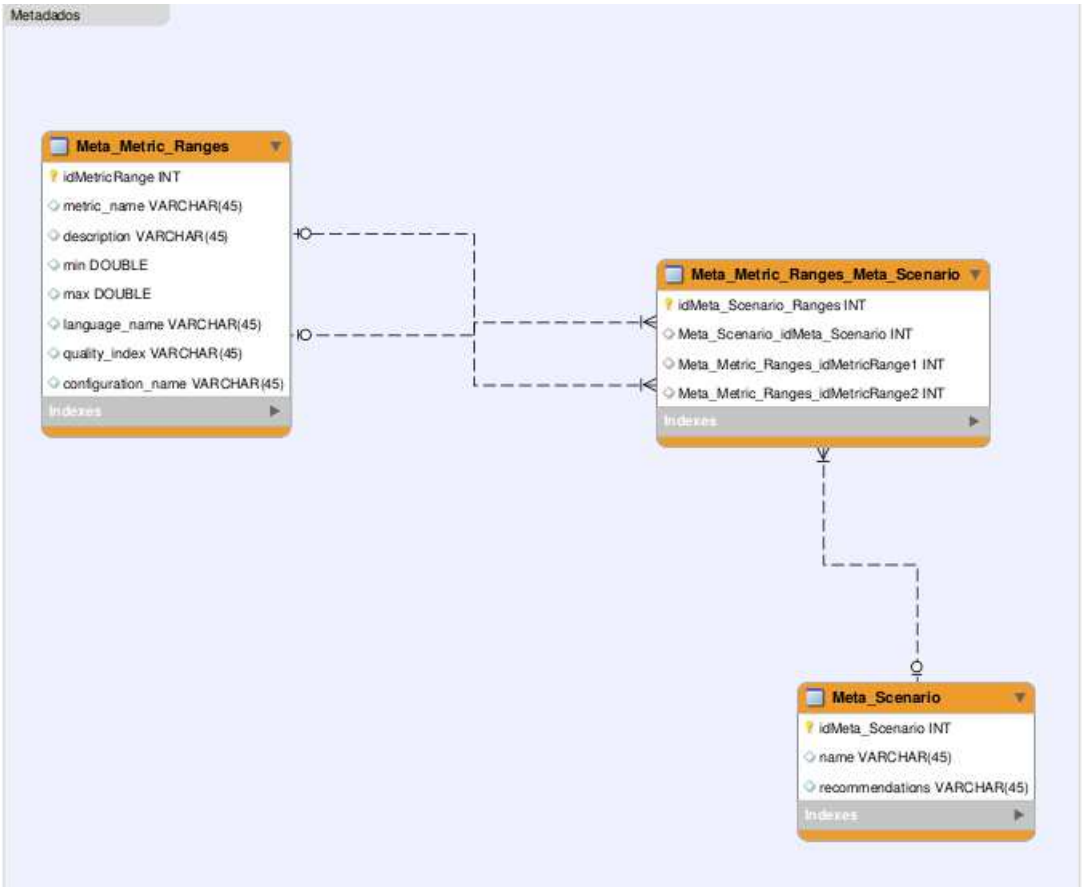

Figura 10 – Projeto físico do *Data Warehouse* extraído de Rêgo (2014)

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

Neste capítulo será apresentada a estratégia de pesquisa adotada durante o trabalho, onde será descrito o 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 maneira 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 focar em 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 as variáveis observadas, 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).

Uma vantagem dos estudos de caso encontra-se no fato deles serem mais fáceis de serem planejados, além de possuírem maior caráter realista. Porém, as desvantagens esbarram no fato dos resultados serem mais difíceis de ser generalizados, ou seja, é possível mostrar os efeitos de uma situação típica, porém é necessária maior análise para que se possa generalizar a outras situações (WOHLIN et al., 2012).

A estrutura do protocolo de estudo de caso proposta se baseia em Brereton, Kitchenham e Budgen (2008) e se apresenta dividida nos seguintes tópicos, cada um com seu objetivo específico:

- **Background - Seção 4.2**: Identificar outros estudos acerca do tópico, definir a questão de pesquisa principal e suas proposições derivadas que serão abordado por este estudo.

- **Design - Seção 4.3**: Identificar se o projeto de pesquisa é um caso único ou múltiplo bem como seu propósito geral.

- **Seleção - Seção 4.4**: Apresentar critérios para a seleção do caso e letcdescrição do objeto de estudo a ser analisado.

- **Fonte e Método de Coleta de Dados - Seção 4.5**: Identificar os dados que serão coletados, definindo um plano para a coleta e como a informação será armazenada.
- **Processo de Análise dos Dados - Seção 4.6**: 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 4.7**: Elicitar tipos de validades aplicáveis a um estudo de caso baseando-se no trabalho desenvolvido por Yin (2001), sendo elas: constructo, interna, externa e confiabiliade.

4.2 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.

4.2.1 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 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 que, para uma determinada linguagem de programação, é possível definir um intervalo qualitativo a partir de um determinado percentil observando o conjunto de valores que são frequentemente observados.

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 foi usado 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 Sommerville (2010), Fenton e Pfleeger (1998), Pandian (2004), Kan (2002), McCabe (1976), Pressman (2010). Outros trabalhos que possuem como tema *data warehousing* foram utilizados durante a revisão bibliográfica, entre eles estão Inmon (2002), Kimball e Ross (2002), Chaudhuri e Dayal (1997).

4.2.2 Questão de Pesquisa

Toda pesquisa se inicia com algum tipo de problema, porém nem todo problema é passível de tratamento científico. Para que um problema seja de natureza científica ele deve estar envolvido a variáveis que podem ser testadas (GIL, 2002). Por causa disso, Gil (2002) ainda define que todo problema deve ser empírico e suscetível de solução, além de ser delimitado a uma dimensão viável.

A questão de pesquisa deriva do problema identificado e busca definir o propósito da medição e o objeto que será medido, seja ele produto, processo ou recurso. Além disso, deve caracterizar o assunto a ser tratado e o ponto de vista que se delimita o escopo (BASILI; CALDIERA; ROMBACH, 1996) (CALDIERA; ROMBACH, 1994).

Para atender a questão de pesquisa foi utilizado o mecanismo goal-question-metrics (GQM). 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). O GQM possui objetivos específicos para a questão de pesquisa, cada um contendo questões específicas que buscam responder através da coleta de métricas a questão geral de pesquisa e assim atender o problema do estudo de caso (BASILI; CALDIERA; ROMBACH, 1996). A estrutura do estudo de caso englobando o GQM utilizado é apresentada da seguinte maneira:

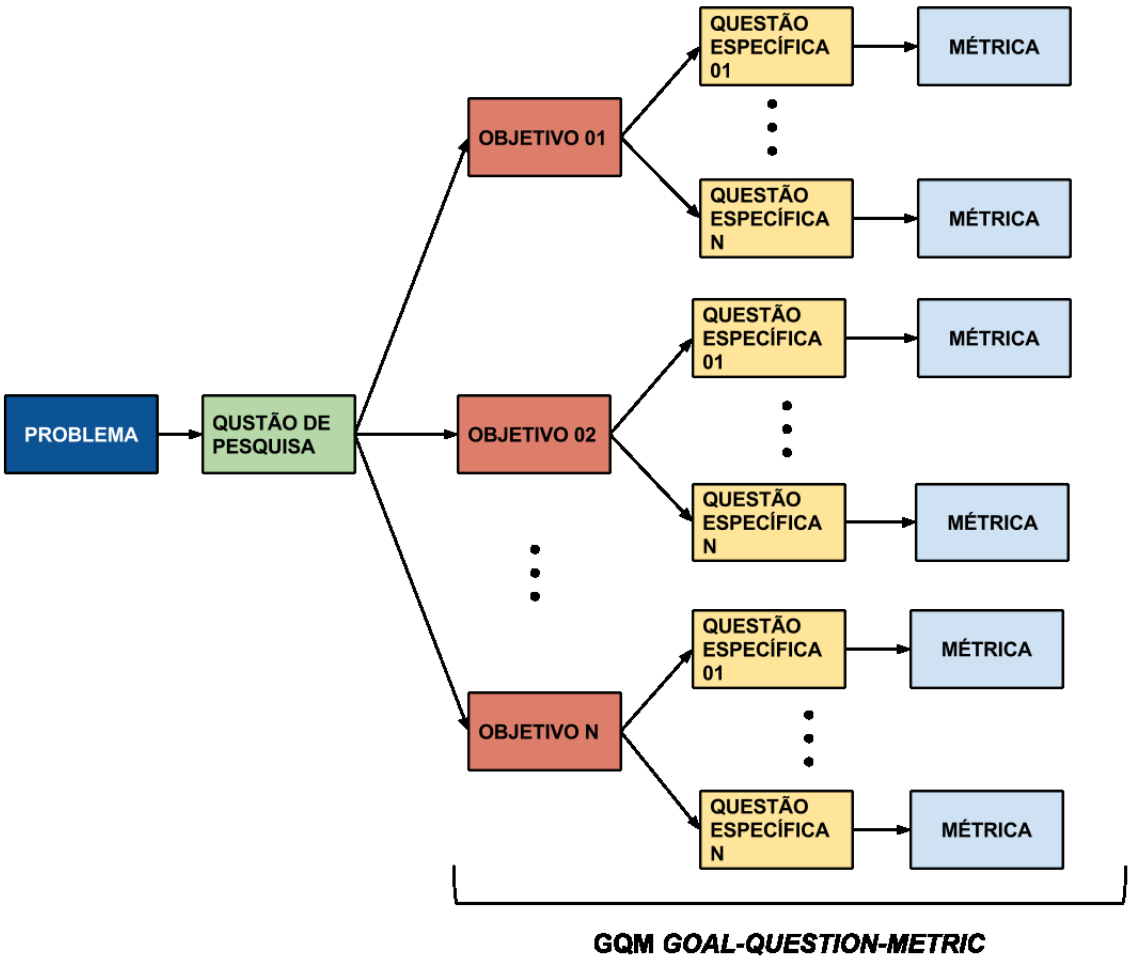

Figura 11 – Estrutura do estudo de caso

Aplicando o modelo estrutural adotado e levando em conta a análise do problema identificado na seção 1.2 foi elaborada a seguinte questão de pesquisa:

Questão de Pesquisa: ***O uso de um ambiente de Data Warehousing para aferição e monitoramento da qualidade interna do código-fonte é eficaz e eficiente do ponto de vista da equipe de qualidade, no contexto do desenvolvimento de projetos internos do TCU?***

Os objetivos identificados para responder a questão geral de pesquisa, bem como suas questões específicas e as métricas levantadas para cada questão serão apresentadas a seguir.

Objetivo 01: Avaliar a eficácia e eficiência do uso de *Data Warehouse* para mo-

nitoramento 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: O intervalo qualitativo que definirá a interpretação desta será desenvolvido a partir de uma análise estatística a ser realizada na segunda parte desse trabalho, analisando valores coletados na aferição dessa métrica em projetos de software livre previamente selecionados para servirem de referência.

4.3 Design

Stake (1995) identifica três modalidades de estudo de caso: intrínseco, instrumental e coletivo. Dentre essas definições, este estudo de caso se caracteriza como instrumental, pois se busca um entendimento geral sobre o problema de pesquisa a partir do estudo de um caso particular. Tal característica se assemelha ao tipo de estudo de caso definido como exploratório por Yin (2001), onde a visão acurada de um caso particular poderá fornecer uma visão geral do problema considerado.

A justificativa para este estudo de um caso particular se dá pelo motivo de que o tema abordado foi pouco explorado até o momento. Assim espera-se que este caso único componha um estudo de múltiplos casos, pois segundo Gil (2002) esta abordagem pode inserir as evidências coletadas nesse trabalho em diferentes contextos, com o intuito de elaborar uma pesquisa de melhor qualidade.

4.4 Seleção

A organização selecionada para este estudo de caso foi, o órgão público, Tribunal de Contas da União-TCU. Para aferir a qualidade de sistemas desenvolvidos internamente no TCU, ou seja, aquele que não é contratado, a equipe responsável pela qualidade estipulou valores para os indicadores das métricas exportadas pela ferramenta Sonar de acordo com pesquisas bibliográficas e pelo próprio consenso entre as equipes de desenvolvimento. Para tanto, essa equipe analisou tanto projetos desenvolvidos anteriores ao uso da prática de monitoramento e melhoria contínua da qualidade interna do código-fonte, quanto os projetos mais recentes, que passaram a fazer uso dessa prática no seu ciclo de desenvolvimento. A interpretação e visualização de dados extraídos da ferramenta *Sonar* são feitas por meio do *plugin Total Quality*. Esse indicador é composto por diferentes tipos de medida/cálculos, sendo os resultados apresentados em um *dashboard* que permite visualizar os valores para as métricas de diferentes formas, entre elas destacam-se rankings e acompanhamento histórico. Cabe ressaltar que, a prática de monitoramento da qualidade do código-fonte no TCU ainda não pode ser considerada institucionalizada, uma vez que, esta prática não é realizada de forma sistemática por todas as equipe de desenvolvimento interno do referido órgão. As dificuldades encontradas pelas equipes de desenvolvimento no TCU estão concentradas no fato de não haver consenso em relação a atualização dos indicadores das métricas. Além disso, após atualizações na ferramenta de análise estática *Sonar*, algumas métricas deixaram de ser medidas, o que tem causado inconsistências na apresentação dos dados relacionados à qualidade do código fonte no *dashboard*.

Portanto, a seleção justifica-se pelo fato do TCU possuir um ambiente onde o estudo da eficácia e eficiência da solução de *data warehousing* pode complementar o monitoramento de métricas de código-fonte já realizado no local.

4.5 Fonte dos Dados Coletados e Método de Coleta

Neste estudo de caso, os dados serão coletados através de registros de observações em campo, questionários e resultados gerados pela própria solução a ser investigada, após análise direta do código fonte.

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

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 solução em análise, como gráficos automaticamente gerados, serão coletados a medida que a ferramenta for utilizada ao longo do tempo.

4.6 Processo de Análise dos Dados

Análise dos dados coletados durante o estudo de caso a ser realizado no TCU 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 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). O uso do GQM mitiga a validade de constructo, 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 pelos medidas extraídas do código-fonte por meio da solução de *DWing* (explicada no capítulo 3), pela análise de questionários e pelos dados coletados através de entrevistas com as equipes de desenvolvimento de de qualidade do órgão.

- **Validade externa:** Yin (2001) recomenda a replicação do estudo em múltiplos casos, por este 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 no TCU, 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, além da base de dados coletados permitirá a repetibilidade deste estudo e consequentemente alcançará validade relacionada à confiabilidade.

4.8 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 Conclusões e Próximos Passos

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 um ambiente de *data warehousing* para monitoramento de métricas de código fonte. Após a revisão bibliográfica ficou mais evidente a importância de se 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 discutidas.

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

BALLARD, C. (Ed.). *Dimensional modeling: in a business intelligence environment*. 1st ed. ed. San Jose, Calif.?: IBM International Technical Support Organization, 2006. (IBM redbooks). ISBN 0738496448. Citado 2 vezes nas páginas 37 e 38.

BASILI, V. R.; CALDIERA, G.; ROMBACH, H. D. *The Goal Question Metric Approach*. [S.l.]: Encyclopedia of Software Engineering, 1996. Citado 2 vezes nas páginas 48 e 49.

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 23.

BECK, K. *Implementation Patterns*. 1. ed. [S.l.]: Addison-Wesley Professional, 2007. Citado 2 vezes nas páginas 15 e 27.

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 2 vezes nas páginas 17 e 46.

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 48 e 49.

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 2 vezes nas páginas 32 e 48.

CHIDAMBER, S. R.; KEMERER, C. F. A metrics suite for object oriented design. *IEEE Trans. Softw. Eng.*, IEEE Press, Piscataway, NJ, USA, v. 20, n. 6, p. 476–493, jun. 1994. ISSN 0098-5589. Disponível em: <http://dx.doi.org/10.1109/32.295895>. Citado na página 23.

CODD, E.; CODD, S.; SALLEY, C. *Providing OLAP (On-line Analytical Processing) to User-analysts: An IT Mandate*. [S.l.]: Codd & Associates, 1993. Citado na página 33.

ELMASRI, R.; NAVATHE, S. B. *Sistemas de banco de dados*. Sao Paulo (SP): Pearson Addison Wesley, 2011. Citado 3 vezes nas páginas 33, 35 e 38.

FENTON, N. E.; PFLEEGER, S. L. *Software Metrics: A Rigorous and Practical Approach*. 2 edition. ed. [S.l.]: Course Technology, 1998. 656 p. Citado 4 vezes nas páginas 14, 20, 21 e 48.

GIL, A. C. *Como elaborar projetos de pesquisa*. [S.l.]: Atlas, 2002. ISBN 9788522431694 8522431698. Citado 3 vezes nas páginas 17, 48 e 51.

GILMORE, W. J. *Dominando PHP e MySql*. [S.l.]: STARLIN ALTA CONSULT, 2008. Citado na página 23.

HITZ, M.; MONTAZERI, B. Measuring Coupling and Cohesion in Object-Oriented Systems. In: *Proceedings of International Symposium on Applied Corporate Computing*. [S.l.: s.n.], 1995. Citado na página 23.

INMON, W. H. *Building the Data Warehouse*. 3rd. ed. New York, NY, USA: John Wiley & Sons, Inc., 2002. Citado 2 vezes nas páginas 32 e 48.

ISO/IEC 15939. *ISO/IEC 15939: Software Engineering - Software Measurement Process*. [S.l.], 2002. Citado 2 vezes nas páginas 20 e 21.

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, 14 e 22.

ISO/IEC 9126. *ISO/IEC 9126-1: Software Engineering - Product Quality*. [S.l.], 2001. Citado 2 vezes nas páginas 20 e 21.

KAN, S. H. *Metrics and models in software quality engineering*. [S.l.]: Addison Wesley, 2002. Citado 5 vezes nas páginas 20, 21, 22, 24 e 48.

KIMBALL, R. *The data warehouse lifecycle toolkit: expert methods for designing, developing, and deploying data warehouses*. [S.l.]: Wiley. com, 1998. Citado 2 vezes nas páginas 36 e 37.

KIMBALL, R. (Ed.). *The data warehouse lifecycle toolkit*. 2nd ed. ed. Indianapolis, IN: Wiley Pub, 2008. Citado na página 33.

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 9 vezes nas páginas 9, 32, 34, 35, 36, 38, 40, 44 e 48.

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 23 e 24.

MACHINI, J. a. et al. *Código Limpo e seu Mapeamento para Métricas de Código-Fonte*. [S.l.]: Universidade de São Paulo, 2010. Citado 6 vezes nas páginas 10, 15, 27, 28, 29 e 48.

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 14 e 47.

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 15.

MCCABE, T. J. A Complexity Measure. *IEEE Transactions Software Engineering*, v. 2, n. 4, p. 308–320, December 1976. Citado 2 vezes nas páginas 22 e 48.

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 10 vezes nas páginas 10, 14, 20, 21, 22, 23, 24, 25, 47 e 55.

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, 33, 34 e 37.

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 48.

PANDIAN, C. R. *Software metrics: a guide to planning, analysis, and application*. Boca Raton, Fla: Auerbach Publications, 2004. ISBN 0849316618. Citado 3 vezes nas páginas 14, 20 e 48.

PRESSMAN, R. S. *Engenharia de software*. Porto Alegre (RS): AMGH, 2010. Citado 4 vezes nas páginas 14, 23, 24 e 48.

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 18 vezes nas páginas 9, 10, 15, 22, 23, 25, 26, 27, 28, 29, 30, 39, 40, 42, 43, 44, 45 e 47.

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 48.

SCHRAMM, W. Notes on case studies of instructional media projects. 1971. Disponível em: <http://eric.ed.gov/?id=ED092145>. Citado na página 46.

SHARMA, N. *Getting started with data warehousing*. [S.l.]: IBM Redbooks, 2011. Citado 6 vezes nas páginas 9, 32, 33, 35, 37 e 38.

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 16 e 17.

SOMMERVILLE, I. *Software Engineering*. 9. ed. Harlow, England: Addison-Wesley, 2010. ISBN 978-0-13-703515-1. Citado na página 48.

STAKE, R. E. *The art of case study research*. Thousand Oaks: Sage Publications, 1995. ISBN 0803957661. Citado na página 51.

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, 37 e 39.

WOHLIN, C. et al. *Experimentation in software engineering*. [S.l.]: Springer, 2012. Citado na página 46.

YIN, R. *Estudo de caso: planejamento e métodos*. [S.l.]: Bookman, 2001. Citado 6 vezes nas páginas 17, 46, 47, 51, 53 e 54.