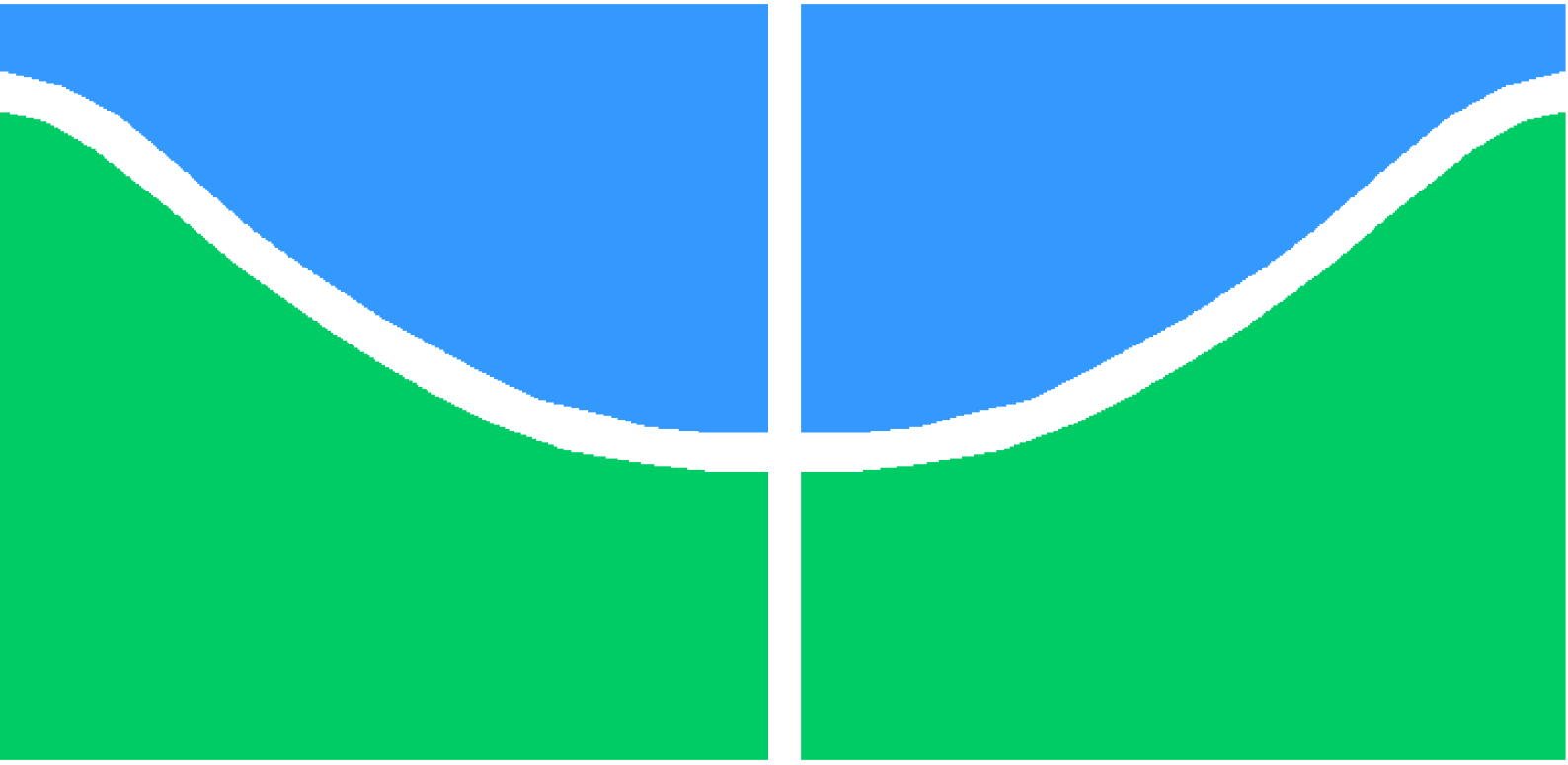

Universidade de Brasília - UnB

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 no TST

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 no TST

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 no TST/ Matheus Oliveira Tristão dos Anjos. – Brasília, DF, 2014-

40 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 no TST

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 no TST

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

Resumo

Esse trabalho tem como objetivo analisar a qualidade do produto e do serviço prestado pela empresa terceirizada, mostrando como o uso de metodologias ágeis e do lean no vínculo contratual podem incluenciar no resultado final

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

Abstract

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*
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

1.1 Contexto

Medir a qualidade do código-fonte de um software é um processo fundamental no desenvolvimento de um software, 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).

O processo de medição, porém, ganha sentido apenas quando há interpretação dos resultados. Usaremos nesse trabalho uma solução desenvolvida por Rêgo (2014) cujo objetivo é atender essa proposta. Para isso, Rêgo (2014) elaborou a seguinte questão de pesquisa:

Como aumentar a visibilidade e facilitar interpretação das métricas de código-fonte a fim de apoiar a decisão de refatoração do ponto de vista de uma equipe de desenvolvimento?

Rêgo (2014) desenvolveu um ambiente de Data Warehousing para armazenamento das métricas de código-fonte extraídas do uso da ferramenta de análise automatizada Analizo buscando atender essa questão de pesquisa. Essa solução tem como objetivo facilitar a interpretação das métricas de código fonte e avaliar indicadores de código limpo no projeto, entre outros objetivos específicos.

1.2 Justificativa

Valores absolutos de métricas não dizem respeito a nada. Solução proposta por Baufaker pode solucionar a interpretação desse valores, automatizando a coleta dos cenários porém

1.3 Problema

Não há uma definiçao concreta a respeito da eficácia e eficiência da solução proposta pelo aluno Rêgo (2014) para monitoramento das métricas de código-fonte no órgão X sendo possível uma substituição do uso do Sonar por essa solução.

1.4 Objetivos

1.5 Hipótese ou Metodologia de pesquisa

1.6 Organização do Trabalho

2 Métricas de Software

Esse capítulo será responsável pela explanação (COMPLETAR)

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. 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 abaixo 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:

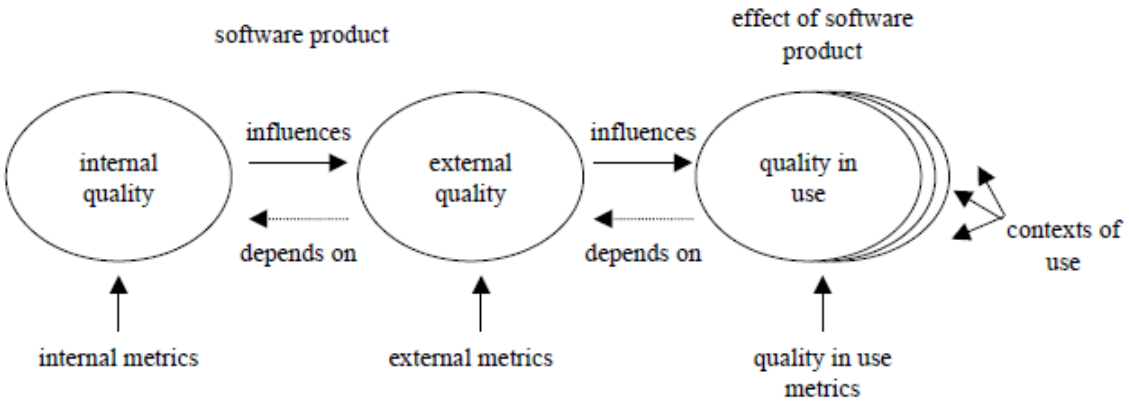

Figura 1 – Modelo de Qualidade do Produto da ISO 25023 adaptado da ISO/IEC 25023 (2011)

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 seguinte tabela 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 seguinte tabela:

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, 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 a seguir 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 vista na tabela abaixo:

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)

3 *Data Warehouse*

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.

3.1 Ciclo de vida de um *Data Warehousing*

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 dados orientados a assunto, podemos entender que... . 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.

3.1.1 *Extraction-Transformation-Load*

Para alcançar as características descritas, 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 descreve a arquitetura de um ambiente de *Data Warehousing*, envolvendo os três processos citados anteriormente

3.2 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 2 descreve:

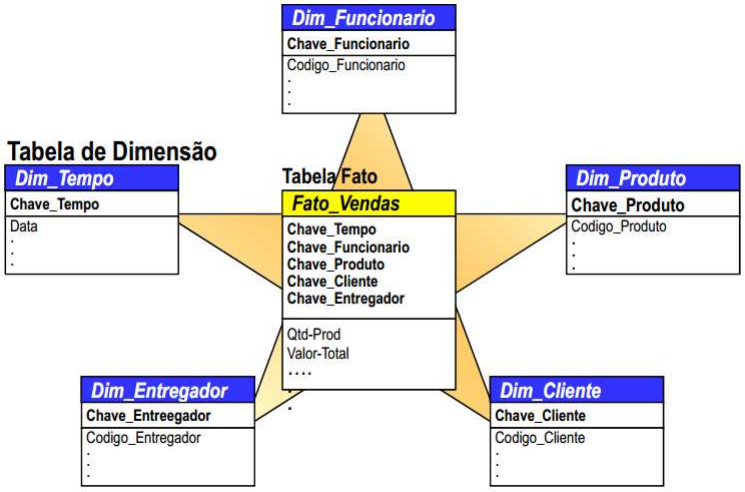

Figura 2 – Esquema estrela adaptado de **??**)

3.2.1 Consultas OLAP

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

5 Projeto de estudo de Caso

Esse capítulo irá tratar da estratégia de pesquisa adotada durante o trabalho, buscando estar de acordo com ...

5.1 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, Schramm (1971) define que 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).

5.2 Modelagem do estudo de caso

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?
- Qual o método de coleta de dados?

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

5.3 Problema

O PROBLEMA

5.4 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, a seguinte questão de pesquisa foi criada após análise do problema:

(ESCREVER QUESTÃO DE PESQUISA)

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).

5.4.1 Objetivos, questões e métricas

Objetivo 01: (ESCREVER)

Questão específica 01: (ESCREVER)

Fonte: (ESCREVER)

Métrica: (ESCREVER)

Questão específica 02: (ESCREVER)

Fonte: (ESCREVER)

Métrica: (ESCREVER)

Objetivo 02: (ESCREVER)

Questão específica 03: (ESCREVER)

Fonte: (ESCREVER)

Métrica: (ESCREVER)

Questão específica 04: (ESCREVER)

Fonte: (ESCREVER)

Métrica: (ESCREVER)

5.5 Fonte dos dados coletados

Os dados foram coletados no TST porque...

5.6 Método de coleta dos dados

O dados foram coletados via análise do código fonte e questionários que...

5.7 Conclusão do capítulo

6 Conclusão

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, 24, 25 e 26.

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 19.

BECK, K. *Implementation Patterns*. 1. ed. [S.l.]: Addison-Wesley Professional, 2007. Citado na página 24.

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 na página 33.

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 28.

FENTON, N. E.; PFLEEGER, S. L. *Software Metrics: A Rigorous and Practical Approach*. 2 edition. ed. [S.l.]: Course Technology, 1998. 656 p. Citado 2 vezes nas páginas 15 e 18.

GILMORE, W. J. *Dominando PHP e MySql*. [S.l.]: STARLIN ALTA CONSULT, 2008. Citado na página 19.

INMON, W. H. *Building the Data Warehouse*. 3rd. ed. New York, NY, USA: John Wiley & Sons, Inc., 2002. Citado na página 28.

ISO/IEC 15939. *ISO/IEC 15939: Software Engineering - Software Measurement Process*. [S.l.], 2002. Citado na página 17.

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 2 vezes nas páginas 9 e 18.

ISO/IEC 9126. *ISO/IEC 9126-1: Software Engineering - Product Quality*. [S.l.], 2001. Citado 2 vezes nas páginas 17 e 18.

KAN, S. H. *Metrics and models in software quality engineering*. [S.l.]: Addison Wesley, 2002. Citado 2 vezes nas páginas 19 e 20.

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 2 vezes nas páginas 28 e 29.

LAIRD, M. C. B. L. M. *Software measurement and estimation: A practical approach*. [S.l.]: Wiley-IEEE Computer Society Press, 2006. Citado na página 20.

MCCABE, T. J. A Complexity Measure. *IEEE Transactions Software Engineering*, v. 2, n. 4, p. 308–320, December 1976. Citado na página 19.

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, 17, 18, 19, 20, 21 e 22.

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 9 vezes nas páginas 10, 15, 19, 22, 23, 24, 25, 26 e 27.

SCHRAMM, W. Notes on case studies of instructional media projects. 1971. Disponível em: <http://eric.ed.gov/?id=ED092145>. Citado na página 32.

SHARMA, N. *Getting started with data warehousing*. [S.l.]: IBM Redbooks, 2011. Citado 2 vezes nas páginas 28 e 29.

TIMES, V. C. *Sistemas de DW*. 2012. Universidade Federal de Pernambuco - UFPE. Disponível em: <www.cin.ufpe.br/~if695/bda_dw.pdf>. Citado na página 30.

YIN, R. *Estudo de caso: planejamento e métodos*. [S.l.]: Bookman, 2001. Citado na página 32.

APÊNDICE A – Descrição do Processo de ETL no Kettle

APÊNDICE B – Gráficos e Tabelas dos Percentis de Métricas de Código-Fonte

APÊNDICE C – Cenários de Limpeza de Código-Fonte