



Universidade de Brasília - UnB
Faculdade UnB Gama - FGA
Engenharia de Software

**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**

Autor: Guilherme Baufaker Rêgo
Orientador: Prof. Msc. Hilmer Rodrigues Neri

Brasília, DF
2014



Guilherme Baufaker Rêgo

**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**

Monografia submetida ao curso de graduação
em Engenharia de Software da Universidade
de Brasília, como requisito parcial para ob-
tenção 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

Guilherme Baufaker Rêgo

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/ Guilherme Baufaker Rêgo. – Brasília, DF, 2014-

123 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. 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

CDU 02:141:005.6

Guilherme Baufaker Rêgo

**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**

Monografia submetida ao curso de graduação
em Engenharia de Software da Universidade
de Brasília, como requisito parcial para ob-
tenção do Título de Bacharel em Engenharia
de Software.

Trabalho aprovado. Brasília, DF, 26 de Junho de 2014:

Prof. Msc. Hilmer Rodrigues Neri
Orientador

**Prof. Dr. Rodrigo Bonifácio de
Almeida**
Convidado 1

**Msc. Débora Reinaldo Arnoud Lima
Formiga**
Convidado 2

Brasília, DF
2014

Este trabalho é dedicado a minha avó paterna, Isaura da Silva, e minha bisavó materna, Enedina Gaspar de Sousa Araújo. Estas foram os exemplos de meus exemplos.

Agradecimentos

Agradeço ao meu orientador Prof. Hilmer Neri por ter sido um professor que estendeu o meu aprendizado sobre o desenvolvimento de software a muito além da sala de aula. A ele sou grato pelas oportunidades concedidas no projeto Desafio Positivo e no SRA, quanto pela orientação no presente trabalho.

Agradeço a Deus, inteligência suprema criadora de todas as coisas, por cada dia que é uma nova chance no caminho da evolução. Também aos meus pais, Juno Rego e Andrea Sousa Araújo Baufaker, pelo dom da vida, pela paciência, pela compreensão, pelo apoio incondicional e sobretudo por mostrar que a educação é um dos únicos bens duráveis e incomensuráveis da vida. Agradeço ainda à Oracina de Sousa Araújo, minha avó materna, que sempre me proporcionou conversas muito bem humoradas sobre a forma de ver o mundo, a vida e o ser humano. Além da maior parte de minha criação, devo a ela desde todo o suporte fornecido em casa até a preocupação sobre a quantidade de horas que estava dormindo durante a noite.

Agradeço a Luisa Helena Lemos da Cruz por ser tão compreensiva, paciente, amorosa e dedicada para comigo, sobretudo nos dias que antecederam a entrega deste trabalho. A ela devo todas as transformações positivas de minha vida desde setembro 2012 até então. Sem sombras de dúvidas, ela é minha fonte de dedicação e inspiração para construir o futuro.

Agradeço a Tina Lemos e Valdo Cruz pelo apoio, pelos conselhos e sobretudo por me receber tão bem quanto um filho é recebido em sua própria casa.

Agradeço a todo apoio dos grandes amigos: Gustavo Nobre Dias, Henrique Leão de Sá Menezes, Danilo Ribeiro Tosta e Carlos Henrique Lima Pontes.

Agradeço a Prof. Eneida Gonzales Valdez por ter me incentivado, com meios de uma verdadeira educadora, a enfrentar os desafios pessoais que a disciplina de Desenho Industrial Assistido por Computador me impôs durante as três vezes que a cursei.

Agradeço também aos companheiros de Engenharia de Software: Vinícius Vieira Meneses, Matheus Freire, Renan Costa, Luiz Mattos, Pedro Tomioka, José Pedro Santana, Aline Gonçalves, Alexandre Almeida, Lucas Kanashiro, Fábio Texeira, Thiago Ribeiro, Alessandro Cateano, Gabriel Silva, Thaiane Braga, Maxwell Almeida e Carlos Oliviera.

"Man sacrifices his health in order to make money. Then he sacrifices money to recuperate his health. Then he is so anxious about the future that he doesn't enjoy the present: the result being that he does not live in the present or the future; he lives as if he is never going to die, and then dies having never really lived"
(Dalai Lama)

Resumo

A qualidade interna do produto de software está diretamente ligada as métricas de código-fonte. Estas por sua vez, quando visualizadas apenas em planilhas decorrentes dos resultados de análise estática de código-fonte não apresentam o grau de visibilidade e interpretação necessários para à tomada de decisão técnica em um determinado projeto de software. Neste trabalho, buscou-se, por meio de um ambiente de *Data Warehousing*, facilitar a interpretação e avaliar as métricas de código-fonte e os cenários de limpeza de código a fim de se obter mais facilmente dados que pudessem apoiar à tomada de decisão técnica, como por exemplo, a refatoração de uma determinada classe do Projeto. Para se validar o ambiente, foi avaliado em um estudo de caso, o Sistema Integrado de Gestão e Conhecimento (SIGC) do Instituto do Patrimônio Artístico Nacional (IPHAN). Esta avaliação resultou em 12 intervalos qualitativos para 12 métricas de código-fonte, em 2 configurações de métricas que utilizaram softwares de referência. Outro resultado foi a detecção de 317 cenários de limpeza de código-fonte em 914 classes na última *release* do projeto avaliado.

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

Abstract

The internal quality of software is strictly connected with source code metrics. When these metrics are seen on spreadsheets which is the outcome from static code analysis, it does not show the sufficient visibility level for making technical decisions in a software project. At this work, a Data Warehousing environment was used in order to gather data related to source code metrics for supporting the decision making process. In spite of validation of Data Warehousing environment, it was applied on a case study which has evaluated the "Sistema Integrado de Gestão e Conhecimento", an information system from Instituto do Patrimônio Artístico Nacional (IPHAN), a Brazilian Department responsible for protecting cultural assets. The evaluation on case study has resulted in 12 quality intervals for source code metrics on 2 different configurations which were made from different softwares references. Additionally, it was found 317 code cleanup opportunities in 914 classes at last release of that software project.

Keywords: *Source Code Metrics. Data Warehousing. Data Warehouse*

Lista de ilustrações

Figura 1 – Modelo de Informação da ISO 15939	22
Figura 2 – Modelo de Qualidade do Produto da ISO 25023 adaptado da ISO/IEC 25023 (2011)	23
Figura 3 – Arquitetura de um ambiente de <i>Data Warehousing</i>	35
Figura 4 – Exemplo de Esquema Estrela adaptado de Times (2012)	37
Figura 5 – Exemplo de Cubo de Dados	37
Figura 6 – Metodologia de Projeto de <i>Data Warehouse</i> proposta por Kimball e Ross (2002)	38
Figura 7 – Diagrama Dimensional de Árvore adaptado de Golfarelli, Maio e Rizzi (1998) e Sampaio (2007)	39
Figura 8 – Arquitetura do Ambiente de <i>Data Warehousing</i> para Métricas de Código-Fonte	44
Figura 9 – Fonte dos Dados do Processos de Negócio: <i>Layout</i> do Arquivo JSON.	48
Figura 10 – Diagrama de Árvore para Valor Percentil das Métricas de Código-Fonte	52
Figura 11 – Diagrama de Árvore para Avaliação de Cenários de Limpeza de Código-Fonte	52
Figura 12 – Diagrama de Árvore para avaliação da Taxa de Aproveitamento de Oportunidade de Melhoria de Código-Fonte	53
Figura 13 – Projeto Físico do <i>Data Warehouse</i>	54
Figura 14 – Metadados do <i>Data Warehouse</i>	55
Figura 15 – Arquitetura Ambiente de <i>Data Warehousing</i> para Métricas de Código-Fonte	59
Figura 16 – Metodologia de Estudo de Caso proposta por Wohlin et al. (2012)	60
Figura 17 – Informações do Repositório de Código-Fonte	67
Figura 18 – Interpretação dos valores percentis conforme a configuração "OpenJDK8 Metrics"	68
Figura 19 – Interpretação dos valores percentis conforme a configuração "Tomcat Metrics"	69
Figura 20 – Total de Cenários de Limpeza de Código-Fonte identificados por cenário e <i>Release</i>	71
Figura 21 – Total de Cenários de Limpeza de Código-Fonte por Release	71
Figura 22 – As 10 classes com menor número de Cénarios de Limpeza	72
Figura 23 – As 10 classes com maior número de Cénarios de Limpeza	72
Figura 24 – Taxa de Aproveitamento de Oportunidades de Melhoria de Código-Fonte	73
Figura 25 – Gráfico da Taxa de Aproveitamento de Oportunidades de Melhoria de Código-Fonte	73

Figura 26 – Crescimento do número de Classes do Projeto SICG	74
Figura 27 – Componentes do Kettle que foram utilizadas nas Transformações	87
Figura 28 – Primeira Transformação realizada no Kettle	87
Figura 29 – Segunda Transformação realizada no Kettle	88
Figura 30 – Terceira Transformação realizada no Kettle	99
Figura 31 – Componentes do Kettle que foram utilizadas nos <i>Jobs</i>	105
Figura 32 – <i>Job</i> deste Trabalho	106
Figura 33 – Intepretação dos Valores Percentis da Métrica ACC	108
Figura 34 – Intepretação dos Valores Percentis da Métrica ACCM	109
Figura 35 – Intepretação dos Valores Percentis da Métrica AMLOC	110
Figura 36 – Intepretação dos Valores Percentis da Métrica ANPM	111
Figura 37 – Intepretação dos Valores Percentis da Métrica CBO	112
Figura 38 – Intepretação dos Valores Percentis da Métrica DIT	113
Figura 39 – Intepretação dos Valores Percentis da Métrica LCOM4	114
Figura 40 – Intepretação dos Valores Percentis da Métrica LOC	115
Figura 41 – Intepretação dos Valores Percentis da Métrica NOC	116
Figura 42 – Intepretação dos Valores Percentis da Métrica NOM	117
Figura 43 – Intepretação dos Valores Percentis da Métrica NPA	118
Figura 44 – Intepretação dos Valores Percentis da Métrica RFC	119

Lista de tabelas

Tabela 1 – Objetivos Específicos do Trabalho	19
Tabela 2 – Objetivos Específico OE6	20
Tabela 3 – Modelo de Informação para métricas de código-fonte com base na ISO/IEC 15939 (2002)	22
Tabela 4 – Percentis para métrica NOM extraídos de Meirelles (2013)	27
Tabela 5 – Nome dos Intervalos de Frequência	28
Tabela 6 – Configurações para os Intervalos das Métricas para Java	29
Tabela 7 – Conceitos de Limpeza extraídos de Machini et al. (2010)	31
Tabela 8 – Cenários de Limpeza	33
Tabela 9 – Diferenças entre OLAP e OLTP extraídas de Times (2012), Rocha (2000) e Neri (2002)	39
Tabela 10 – Exemplo do Total de Vendas de uma Rede de Lojas no mês de Novembro	41
Tabela 11 – Exemplo do Total de Vendas de uma rede de lojas no mês de novembro com a dimensão Produto	41
Tabela 12 – Exemplo do Total de vendas da Loja Norte no mês de novembro	41
Tabela 13 – Exemplo de Vendas por produto de uma rede de lojas nos meses de novembro e dezembro	42
Tabela 14 – Exemplo de Vendas do Produto A na rede de Lojas	42
Tabela 15 – Exemplo de Vendas por Loja para cada um dos Produtos nos meses de Novembro e Dezembro	42
Tabela 16 – Critérios Gerais de seleção de ferramentas	45
Tabela 17 – Critérios Específicos para Ferramenta de Análise Estática de Código-Fonte	45
Tabela 18 – Características do SonarQube e do Analizo	46
Tabela 19 – Análise do SonarQube e do Analizo quanto aos critérios gerais e quanto aos critérios específicos de ferramentas de análise estática	47
Tabela 20 – Requisitos de Negócio da Avaliação dos Valores Percentis das Métricas de Código-Fonte conforme as configurações especificadas na Tabela 6	49
Tabela 21 – Requisitos de Negócio da Avaliação de Cenários de Limpeza de Código-Fonte e Avaliação de Taxa de Aproveitamento de Oportunidades de Melhoria de Código-Fonte conforme a Tabela 8	50
Tabela 22 – Fatos e Dimensões do <i>Projeto de Data Warehouse</i>	51
Tabela 23 – Características do Kettle e avaliação quanto aos critérios gerais de seleção de ferramentas	56
Tabela 24 – Características do Pentaho BI Platform e avaliação quanto aos critérios gerais de seleção de ferramentas	57

Tabela 25 – Características do Saiku Analytics e avaliação quanto aos critérios gerais de seleção de ferramentas	57
Tabela 26 – Objetivos Específicos de Estudo de Caso	61
Tabela 27 – Critérios de Seleção de Objeto do Estudo de Caso	61
Tabela 28 – Dados a serem analisados no Estudo de Caso	65
Tabela 29 – Objetivos Específicos do Trabalho	75
Tabela 30 – Comparação entre este trabalho e o relatório de auditoria de código-fonte	76
Tabela 31 – Tempo de Execução de cada passo automatizado no processo de medição de métricas de código-fonte	77
Tabela 32 – Total de Consultas OLAP realizadas	78
Tabela 33 – Classes com Cenário de Limpeza: Classe com métodos muito grandes e/ou muitos condicionais	120
Tabela 34 – Classes com Cenário de Limpeza: Classe com muitos filhos	120
Tabela 35 – Classes com Cenário de Limpeza: Classe com muita exposição	120
Tabela 36 – Classes com Cenário de Limpeza: Classe Pouco Coesa	121
Tabela 37 – Classes com Cenário de Limpeza: Complexidade Estrutural	122
Tabela 38 – Classes com Cenário de Limpeza: Interface dos Métodos	123

Lista de abreviaturas e siglas

ACC	<i>Afferent Connections per Class</i>
ACCM	<i>Average Cyclomatic Complexity per Method</i>
AMLOC	<i>Average Method Lines of Code</i>
ANPM	<i>Average Number of Parameters per Method</i>
CBO	<i>Coupling Between Objects</i>
CSV	<i>Comma-Separated Values</i>
DER	Diagrama Entidade Relacionamento
DIT	<i>Depth of Inheritance Tree</i>
DW	<i>Data Warehouse</i>
ETL	<i>Extraction-Transformation-Load</i>
FTP	<i>File Transfer Protocol</i>
GQM	<i>Goal-Question-Metric</i>
IEC	<i>International Electrotechnical Commission</i>
IPHAN	Instituto do Patrimônio Histórico e Artístico Nacional
ISO	<i>International Organization for Standardization</i>
JSON	<i>JavaScript Object Notation</i>
LCOM4	<i>Lack of Cohesion in Methods</i>
LOC	<i>Lines of Code</i>
NPA	<i>Number of Public Attributes</i>
NOC	<i>Number of Children</i>
NOM	<i>Number of Methods</i>
OLAP	<i>On-Line Analytical Processing</i>
OLTP	<i>Online Transaction Processing</i>

RFC	<i>Response For a Class</i>
SCAM	<i>IEEE International Working Conference on Source Code Analysis and Manipulation</i>
SGBD	Sistema de Gerenciamento de Bancos de Dados
SICG	Sistema Integrado de Conhecimento e Gestão
XML	<i>Extensible Markup Language</i>
YAML	<i>YAML Ain't Markup Language</i>

Sumário

1	INTRODUÇÃO	17
1.1	Contexto	17
1.2	Problema	17
1.3	Questão de Pesquisa	17
1.4	Objetivos	18
1.5	Hipótese	19
1.6	Organização do Trabalho	20
2	MÉTRICAS DE SOFTWARE	21
2.1	Processo de Medição de Software	21
2.2	Classificação das Métricas de Software	22
2.3	Métricas de Código-Fonte	24
2.3.1	Métrica de Tamanho e Complexidade	24
2.3.2	Métricas de Orientação à Objetos	25
2.3.3	Configurações de Qualidade para Métricas de Código-Fonte	26
2.3.4	Código Limpo	30
3	DATA WAREHOUSING	34
3.1	<i>Extraction-Transformation-Load (ETL)</i>	35
3.2	<i>Data Warehouse</i>	36
3.2.1	Metodologia do Projeto do <i>Data Warehouse</i>	38
3.3	<i>On-Line Analytical Processing (OLAP)</i>	39
3.4	Visualização de Dados	42
4	AMBIENTE DE DATA WAREHOUSING PARA MÉTRICAS DE CÓDIGO-FONTE	44
4.1	Arquitetura do Ambiente <i>Data Warehousing</i> para Métricas de Código-Fonte	44
4.2	Ferramenta de Análise Estática de Código-Fonte	45
4.3	Projeto do <i>Data Warehouse</i>	47
4.4	Ferramentas de <i>Data Warehousing</i>	55
4.4.1	Implementação da Extração, Transformação e Carga dos Dados	56
4.4.2	Implementação das Consultas OLAP e Visualização de Dados	56
4.5	Resumo das Ferramentas utilizadas do Ambiente de <i>Data Warehousing</i>	59

5	ESTUDO DE CASO	60
5.1	Planejamento do Estudo de Caso	60
5.1.1	Trabalhos Relacionados ao Estudo de Caso	60
5.1.2	Objetivos do Estudo de Caso	60
5.1.3	Seleção de Objeto do Estudo de Caso	61
5.1.4	Objeto de Estudo de Caso	62
5.1.4.1	Visão Geral dos Requisitos	62
5.1.4.2	O Software Analisado	63
5.1.5	Dados, Fonte dos Dados e Forma de Análise dos Dados	64
5.1.6	Ameaças à Validade do Estudo de Caso	65
5.2	Execução do Estudo de Caso e Análise dos Dados	66
5.2.1	Análise dos Valores Percentis para Métricas de Código-Fonte do SICG	67
5.2.2	Análise dos Cenários de Limpeza identificados no SICG	71
5.2.3	Análise da Taxa de Aproveitamento de Oportunidade de Melhoria de Código-Fonte	72
6	CONCLUSÃO	75
6.1	Limitações	79
6.2	Trabalhos Futuros	79
	Referências	81
	APÊNDICE A – DESCRIÇÃO DO PROCESSO DE ETL NO KETTLE	86
A.1	Implementações das <i>Transformations</i>	86
A.2	Implementação do <i>Job</i>	105
	APÊNDICE B – GRÁFICOS E TABELAS DOS PERCENTIS DE MÉTRICAS DE CÓDIGO-FONTE	107
	APÊNDICE C – CENÁRIOS DE LIMPEZA DE CÓDIGO-FONTE	120

1 Introdução

1.1 Contexto

A qualidade do software depende da qualidade do código-fonte, pois um bom código-fonte é um bom indicador de qualidade interna do produto de software (BECK, 2003) (ISO/IEC 25023, 2011). Portanto, decisões errôneas de desenvolvedores podem gerar, por exemplo, trechos de código não coesos, que venham a "degradar" com o tempo e que aumentam exponencialmente a chance de manutenções corretivas onerosas (BECK, 2007) (BECK, 1999b).

Entre as formas de analisar a qualidade do código-fonte, há a análise estática de código-fonte que é uma análise automatizada das estruturas internas do código, que permite obter métricas de código-fonte (EMANUELSSON; NILSSON, 2008) (WICHMANN et al., 1995) (NIELSON; NIELSON; HANKIN, 1999) (SOMMERVILLE, 2010). Alguns trabalhos como Marinescu e Ratiu (2004), Marinescu (2005), Moha, Guéhéneuc e Leduc (2006), Moha et al. (2008), Moha et al. (2010) e Rao e Reddy (2007) mostraram que é possível, a partir da análise de métricas de código-fonte, obter indicadores de pedaços não coesos e com alto acoplamento. Estes trechos são passíveis de eliminação com a refatoração, que é a técnica de modificação das estruturas internas do código-fonte sem modificação do comportamento externo observável (FOWLER, 1999).

1.2 Problema

Embora a refatoração seja uma prática bastante documentada e utilizada principalmente nos processos de desenvolvimento que utilizam métodos ágeis (BECK, 1999b), a decisão de aplicação, quer seja em uma classe, módulo ou projeto, envolve a avaliação de fatores como custo, prazo, risco e qualidade do código-fonte (YAMASHITA, 2013).

Este último fator, que pode ser determinante para decisão de refatorar, normalmente, é difícil de ser avaliado. Isso ocorre, pois as ferramentas de análise estática de código-fonte, normalmente, não apresentam associação entre as medidas numéricas e a forma de interpretá-los, isto é, as métricas frequentemente mostram apenas valores numéricos isolados (MEIRELLES, 2013). Além disso, os resultados obtidos a partir de ferramentas de análise estática de código-fonte são apresentados em longos arquivos (planilhas, arquivos JSON, XML), ou seja, sem mecanismos de separação, agregação e formas de visualização de dados que permitam a fácil identificação de um ponto de melhoria no código-fonte.

1.3 Questão de Pesquisa

Tendo em vista que este problema afeta a avaliação de projetos de software, pois é crucial que informações sobre o desenvolvimento de software sejam coletadas e compartilhadas entre projetos e pessoas em uma visão organizacional unificada, para que determinada organização possa compreender o processo de medição e monitoramento de projetos de software e, conseqüentemente, se tornar mais hábil e eficiente em realizar atividades técnicas relacionadas ao processo de desenvolvimento de software (CHULANI et al., 2003), foi elaborada 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?

1.4 Objetivos

Colocando a resolução da questão de pesquisa como objetivo geral do trabalho, foram elaborados objetivos específicos utilizando o GQM (BASILI; CALDIERA; ROMBACH, 1996). Nesta abordagem cada objetivo específico derivou uma série de questões específicas que são respondidas com métricas específicas, tal como se mostra na Tabela 2.

Objetivo Específico	Questão Específica	Métricas Específicas
OE1 - Avaliar a Qualidade do Código-Fonte no Projeto	QE1 - Qual conjunto de poucas métricas de código-fonte pode indicar a qualidade do código-fonte em projeto?	M1 - Métricas de Código-Fonte, apresentadas na Seção 2.3.
OE2 - Automatizar o processo de medição de métricas de código-fonte.	QE2 - Como automatizar o processo de medição de métricas de código-fonte?	M2.1 - Quantidade de passos automatizados apresentados no Capítulo 4. M2.2 - Tempo de Execução de cada um dos passos automatizados apresentados no Capítulo 4.
OE3 - Facilitar a interpretação das métricas de código-fonte.	QE3 - Como o conjunto de poucas métricas de código-fonte podem ser melhor interpretadas pela equipe de desenvolvimento?	M3 - Configurações de Intervalos Qualitativos para Métricas de Código-Fonte apresentados na seção 2.3.3.
OE4 - Avaliar indicadores de Código Limpo no Projeto.	QE4 - Quais são os indicadores de código-limpo em um código-fonte de um determinado projeto?	M4 - Quantidade de Cenários de Limpeza, apresentados na Seção 2.3.4.
OE5 - Avaliar indicadores de aproveitamento de oportunidades de melhoria de código-fonte	QE5 - Qual é o aproveitamento de oportunidades de melhoria de código-fonte em um determinado Projeto?	M5 - Taxa de Aproveitamento de Oportunidades de Melhoria de Código.

Tabela 1 – Objetivos Específicos do Trabalho

A Taxa de Aproveitamento de Oportunidades de Melhoria de Código, em uma release, é calculada como:

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

onde Ce é o total de cenários de limpezas identificados e Cl é total de classes em uma release.

1.5 Hipótese

Visando atingir os objetivos específicos estabelecidos na Tabela 2, realizou-se uma revisão bibliográfica da literatura em busca de ambientes de informação que visassem à automação de processos e exposição de informações em âmbito gerencial. Alguns trabalhos, como Palza, Fuhrman e Abran (2003), Ruiz et al. (2005), Castellanos et al. (2005), Becker et al. (2006), Folleco et al. (2007), Silveira, Becker e Ruiz (2010), evidenciaram

que ambientes de *Data Warehousing* são boas soluções para adoção de um programa de métricas em processos de desenvolvimento de software.

Em conformidade com o referencial teórico enunciado anteriormente, foi formulada a hipótese H0:

H0: A construção de um ambiente de *Data Warehousing* traz automação necessária ao processo de medição de métricas de código-fonte, a possibilidade de criação de um campo de conhecimento semântico para facilitar a interpretação de métricas de código-fonte e a visibilidade necessária às métricas de código-fonte, bem como a possibilidade de inferência de outras informações advindas destas.

Tendo em vista a adoção da hipótese, adicionou-se o objetivo OE6:

Objetivo Específico	Questão Específica	Métricas Específicas
OE6 - Avaliar quantidade de consultas OLAP sobre as métricas de código-fonte realizadas no ambiente de <i>Data Warehousing</i> em um projeto.	QE6 - Quantas consultas OLAP são necessárias para atender os requisitos de visualizações das métricas de código-fonte, cenários de limpeza de código-fonte e avaliação de Taxa de Aproveitamento de Oportunidades de Limpeza de Código?	M6 - Quantidade de consultas OLAP realizadas

Tabela 2 – Objetivos Específico OE6

1.6 Organização do Trabalho

Após a leitura da introdução, encontrar-se-á adiante, no Capítulo 2, as métricas de software, processo de medição da ISO 15939, as métricas de código-fonte, código-limpo e os cenários de limpeza de código-fonte; O Capítulo 3 que apresenta conceitualmente o ambiente de *Data Warehousing*. O Capítulo 4 que apresenta o projeto e a implementação do ambiente de *Data Warehousing* para métricas de código-fonte e cenários de código limpo construído neste trabalho. O capítulo 5 apresenta o projeto de estudo de caso do uso como forma de validar a utilização do ambiente. Por fim, o capítulo 6 descreve os resultados obtidos com aplicação do ambiente em forma de estudo de caso, onde as 24 *releases* do software SICG (Sistema Integrado de Conhecimento e Gestão) do Instituto de Patrimônio Histórico e Artístico Nacional (IPHAN), foram analisadas. Adicionalmente no capítulo 6, são discutidos as conclusões, limitações do trabalho e trabalhos futuros.

2 Métricas de Software

2.1 Processo de Medição de Software

Segundo [Fenton e Pfleeger \(1998\)](#), medição é o mapeamento de relações empíricas em relações formais. Isto é, quantificação em símbolos com objetivo de caracterizar uma entidade por meio de seus atributos. Contrapondo-se com uma definição operacional, a [ISO/IEC 15939 \(2002\)](#) define medição como conjunto de operações que visam por meio de um objeto determinar um valor a uma medida ou métrica ¹. Alguns modelos de referência, como [CMMI \(2010\)](#), e até a própria [ISO/IEC 15939 \(2002\)](#) definem medição como uma ferramenta primordial para gerenciar as atividades do desenvolvimento de software e para avaliar a qualidade dos produtos e a capacidade de processos organizacionais.

A [ISO/IEC 15939 \(2002\)](#) define um processo de medição com base em um modelo de informação, que é mostrado na Figura 1, com intuito de obter produtos de informação para cada necessidade de informação. Para isto, cada necessidade de informação, que é uma situação que requer conhecimento de objetivos, metas, riscos e problemas, é mapeada em uma construção mensurável que tem em sua origem um conceito mensurável, como por exemplo, tamanho, qualidade e custo. A partir do conceito mensurável, é possível mapear um atributo, característica que permite distinguir qualitativamente e/ou quantitativamente uma entidade que pode ser um processo, projeto ou produto de software. Por fim, cada construção mensurável é mapeada em um ou mais produtos de informação que levam uma ou mais métricas, que podem ser classificadas sob critérios apresentados na Seção 2.2.

¹ A definição formal da [ISO/IEC 15939 \(2002\)](#) não utiliza o termo métrica. Tendo em vista que o termo métrica é mais difundido em processos de medição e análise, tais como os que se baseiam diretamente ou indiretamente no *Goal-Question-Metric* de [Basili, Caldiera e Rombach \(1996\)](#), resolveu-se adotar o termo métrica com mesmo valor semântico ao termo medida da [ISO/IEC 15939 \(2002\)](#), que é o valor operacional propriamente dito

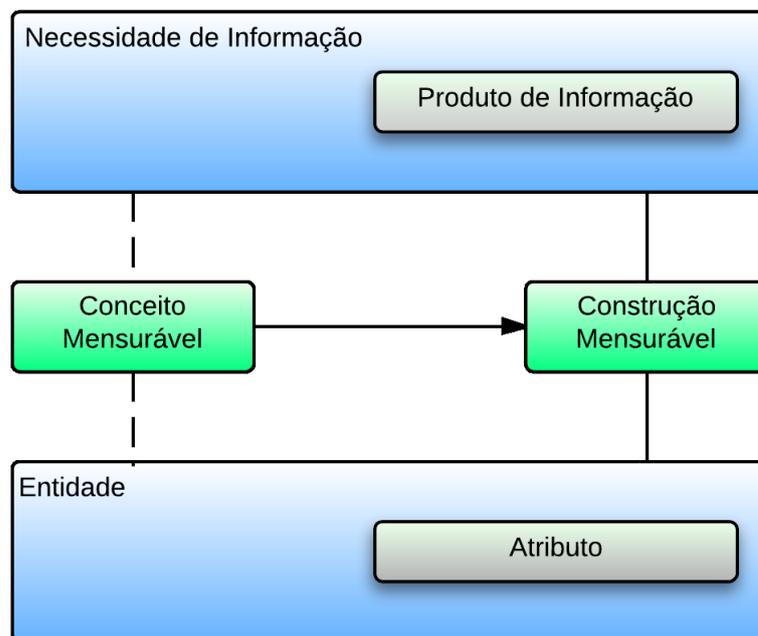


Figura 1 – Modelo de Informação da ISO 15939

O modelo de informação da [ISO/IEC 15939 \(2002\)](#) é utilizado para construir o propósito geral das medições e a identificação de medidas ou métricas que respondem as necessidades de informação. Quando se trata da coletas de métricas de código-fonte, é possível construir um modelo de informação tal como na Tabela 3.

Entidade	Código Fonte
Conceito Mensurável	Qualidade
Construção Mensurável	Qualidade do Código-Fonte
Atributos a ser medidos	Classes, Métodos e Pacotes
Produto de Informação	Indicadores de Qualidade do Código-Fonte

Tabela 3 – Modelo de Informação para métricas de código-fonte com base na [ISO/IEC 15939 \(2002\)](#)

2.2 Classificação das Métricas de Software

As métricas de software possuem uma escala de medição, que é um conjunto ordenado de valores, contínuos ou discretos, ou uma série de categorias nas quais uma entidade é mapeada ([ISO/IEC 15939, 2002](#)). As escalas podem ser:

- **Nominal:** A medição é categórica. Nesta escala, só é possível realização de comparações, sendo que a ordem não possui significado ([ISO/IEC 15939, 2002](#)) ([FENTON; PFLEEGER, 1998](#)) ([MEIRELLES, 2013](#)).

- **Ordinal:** A medição é baseada em ordenação, ou seja, os valores possuem ordem, mas a distância entre eles não possui significado. Por exemplo, nível de experiência dos programadores (ISO/IEC 15939, 2002) (FENTON; PFLEEGER, 1998) (MEIRELLES, 2013).
- **Intervalo:** A medição é baseada em distâncias iguais definidas para as menores unidades. Por exemplo, o aumento de 1° C de um termômetro. Nesta escala é possível realizar ordenação, soma e subtração (ISO/IEC 15939, 2002) (FENTON; PFLEEGER, 1998).
- **Racional:** A medição é baseada em distâncias iguais definidas para as menores unidades, e neste caso é possível a ausência por meio do zero absoluto. Por exemplo, a quantidade de linhas de código em uma classe. Nesta escala, é possível realizar ordenação, soma, subtração, multiplicação e divisão (ISO/IEC 15939, 2002) (FENTON; PFLEEGER, 1998).

As métricas podem ser classificadas quanto ao objeto da métrica, que divide as métricas de software em: *métricas de processo* e *métricas de produto* (MILLS, 1999). Ainda é possível, segundo a ISO/IEC 15939 (2002), dividir as métricas quanto ao método de medição, sendo que estas podem ser *métricas objetivas*, que são baseadas em regras numéricas e podem ter a coleta manual ou automática, ou *métricas subjetivas*, que envolvem o julgamento humano para consolidação do resultado.

Segundo o modelo de qualidade da ISO/IEC 25023 (2011), que é mostrado na Figura 2, as métricas de produto podem ser subdivididas em três categorias:

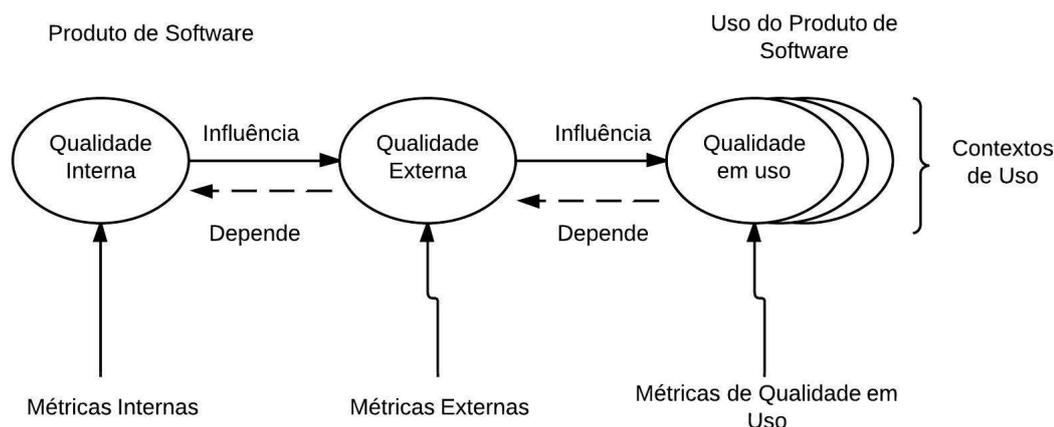


Figura 2 – Modelo de Qualidade do Produto da ISO 25023 adaptado da ISO/IEC 25023 (2011)

- **Métricas Internas:** São métricas que aferem a qualidade interna do software por meio da avaliação de estruturas internas que compõem o software em estágio de

desenvolvimento. São conhecidas como métricas de código-fonte.

- **Métricas Externas:** São métricas que capturam o comportamento do software e só podem ser aferidas por atividades de teste do desenvolvimento do software em condições similares as que serão encontradas em ambientes de implantação.
- **Métricas de Qualidade em Uso:** São métricas que aferem se o software atende as necessidades do cliente com eficiência, produtividade, segurança e satisfação em contextos específicos de uso. Estas só podem ser coletadas em ambientes reais, isto é, o ambiente de produção.

2.3 Métricas de Código-Fonte

Segundo a *International Working Conference on Source Code Analysis and Manipulation* (SCAM), o código-fonte é qualquer especificação executável de um sistema de software. Por conseguinte, se inclui desde código de máquina até linguagens de alto nível ou representações gráficas executáveis (HARMAN, 2010). Este fato implica às métricas de código-fonte as categorias de métricas objetivas e características como validade, simplicidade, objetividade, fácil obtenção e robustez (MILLS, 1999).

2.3.1 Métrica de Tamanho e Complexidade

O tamanho do código-fonte foi um dos primeiros conceitos mensuráveis do software, dado que o software poderia ocupar espaço tanto em forma de cartões perfurados quanto em forma de papel quando o código-fonte era impresso. A segunda lei de Lehman (1980) enuncia que a complexidade aumenta à medida que o software é evoluído. Logo, é perceptível que as métricas de complexidade estão diretamente ligadas as métricas de tamanho, sendo que a modificação em uma provavelmente impactará na outra. A seguir são apresentadas algumas métricas de tamanho e complexidade:

- **LOC** (*Lines of Code* - Número de Linhas de Código) foi uma das primeiras métricas utilizadas para medir o tamanho de um software. São contadas apenas as linhas executáveis, ou seja, são excluídas linhas em branco e comentários. Para efetuar comparações entre sistemas usando LOC, é necessário que ambos tenham sido feitos na mesma linguagem de programação e que o estilo esteja normalizado (JONES, 1991).
- **ACCM** (*Average Cyclomatic Complexity per Method* - Média da Complexidade Ciclomática por Método) mede a complexidade dos métodos ou funções de um programa. Essa métrica pode ser representada através de um grafo de fluxo de controle (MCCABE, 1976). O uso de estruturas de controle, tais como, *if*, *else*, *while* aumentam a complexidade ciclomática de um método.

- **AMLOC** (*Average Method Lines of Code* - Média do número de linhas de código por método) indica se o código está bem distribuído entre os métodos. Quanto maior, mais pesados são os métodos. É preferível ter muitas operações pequenas e de fácil entendimento que poucas operações grandes e complexas (MEIRELLES, 2013).

2.3.2 Métricas de Orientação à Objetos

O paradigma da orientação a objetos permitiu aos programadores abstrair computação ao negócio. Isto significou uma evolução no desenvolvimento quando comparado ao paradigma procedural, fato que implicou a paradigma de orientação à objetos uma longa permanência na academia quanto na indústria de desenvolvimento de software (LI; HENRY, 1993).

Dado a grande utilização do paradigma de orientação à objetos no desenvolvimento de software, foram selecionadas algumas das principais métricas de orientação a objetos:

- **ACC** (*Afferent Connections per Class* - Conexões Aferentes por Classe) é o número total de classes externas de um pacote que dependem de classes de dentro desse pacote. Quando calculada no nível da classe, essa medida também é conhecida como *Fan-in* da classe, medindo o número de classes das quais a classe é derivada e, assim, valores elevados indicam uso excessivo de herança múltipla (MCCABE; DREYER; WATSON, 1994) (CHIDAMBER; KEMERER, 1994).
- **ANPM** (*Average Number of Parameters per Method* - Média do Número de Parâmetros por Método) calcula a média de parâmetros dos métodos da classe. Seu valor mínimo é zero e não existe um limite máximo para o seu resultado, mas um número alto de parâmetros pode indicar que um método pode ter mais de uma responsabilidade (BASILI; ROMBACH, 1987).
- **CBO** (*Coupling Between Objects* - Acoplamento entre Objetos) é o número total de classes dentro de um pacote que dependem de classes externas ao pacote. Quando calculada no nível da classe, essa medida também é conhecida como *Fan-out* da classe (CHIDAMBER; KEMERER, 1994).
- **DIT** (*Depth of Inheritance Tree* - Profundidade da Árvore de Herança) é o número de superclasses ou classes ancestrais da classe sendo analisada. São contabilizadas apenas as superclasses do sistema, ou seja, as classes de bibliotecas não são contabilizadas. Nos casos onde herança múltipla é permitida, considera-se o maior caminho da classe até uma das raízes da hierarquia. Quanto maior for o valor DIT, maior é o número de atributos e métodos herdados, e, portanto, maior é a complexidade (SHIH et al., 1997).

- **LCOM4** (*Lack of Cohesion in Methods* - Falta de Coesão entre Métodos). Originalmente proposto por [Chidamber e Kemerer \(1994\)](#) como LCOM não teve uma grande aceitabilidade. Após críticas e sugestões, a métrica foi revisada por [Hitz e Montazeri \(1995\)](#). Para calcular LCOM4 de um módulo, é necessário construir um gráfico não-orientado em que os nós são os métodos e atributos de uma classe. Para cada método, deve haver uma aresta entre ele e um outro método ou variável que ele usa. O valor da LCOM4 é o número de componentes fracamente conectados nesse gráfico.
- **NOC** (*Number of Children* - Número de Filhos) é o número de subclasses ou classes filhas que herdam da classe analisada ([ROSENBERG; HYATT, 1997](#)). Deve-se ter cautela ao modificar classes com muitos filhos, pois uma simples modificação de assinatura de um método pode criar uma mudança em muitas classes.
- **NOM** (*Number of Methods* - Número de Métodos) é usado para medir o tamanho das classes em termos das suas operações implementadas. Esta métrica pode identificar o potencial de reuso de uma classe. Em geral, as classes com um grande número de métodos são mais difíceis de serem reutilizadas, pois elas são propensas a serem menos coesas ([LORENZ; KIDD, 1994](#)).
- **NPA** (*Number of Public Attributes* - Número de Atributos Públicos) mede o encapsulamento. Os atributos de uma classe devem servir apenas às funcionalidades da própria classe. Portanto, as boas práticas de programação recomendam que os atributos de uma classe devem ser manipulados por meio dos métodos de acesso ([BECK, 1997](#)).
- **RFC** (*Response For a Class* - Respostas para uma Classe) é o número de métodos dentre todos os métodos que podem ser invocados em resposta a uma mensagem enviada por um objeto de uma classe ([SHARBLE; COHEN, 1993](#)).

2.3.3 Configurações de Qualidade para Métricas de Código-Fonte

No trabalho de [Meirelles \(2013\)](#), analisou-se a distribuição estatística de métricas de código-fonte em 38 projetos de software livre com mais de 100.000 downloads, como por exemplo, Tomcat, OpenJDK, Eclipse, Google Chrome, VLC e entre outros. Para tal [Meirelles \(2013\)](#), utilizou-se da técnica de estatística descritiva: percentil. Esta é medida que divide a amostra ordenada (por ordem crescente dos dados) em 100 partes, cada uma com uma percentagem de dados aproximadamente igual.

Define-se percentil k , Q_k , para $k = 1, 2, \dots, 99$, como sendo o valor tal que $k\%$ dos elementos da amostra são menores ou iguais a Q_k e os restantes $(100-k)\%$ elementos da amostra são maiores ou iguais a Q_k . O 25º percentil recebe o nome de primeiro quartil, O

50º percentil de mediana ou segundo quartil e o 75º percentil recebe o nome de terceiro quartil.

Após realizar aplicação da técnica estatística na série de dados das métricas de código-fonte obtidos da análise estática de código-fonte dos projetos de software livre, obteve-se tabelas como a Tabela 4.

Software	Mín	1%	5%	10%	25%	50%	75%	90%	95%	99%	Máx
Linguagem: C											
Linux	1,0	1,0	1,0	2,0	5,0	11,0	24,0	46,0	67,0	136,0	639,0
Freebsd	0,0	0,0	0,0	0,0	1,0	4,0	13,0	29,0	48,0	110,0	1535,0
Android	1	1	1	1	2	4	9	20	31	78	572
Bash	1	1	1	1	2	4	13	34	56	114	185
Chromium OS	1	1	1	1	3	7	16	32	49	104	434
GCC	1	1	1	1	2	2	7	17	30	89	3439
Gimp	1	1	2	2	5	8	16	28	37	79	179
Git	1	1	1	1	3	8	19	41	57	111	321
Gnome	1	1	1	2	4	10	21	38	56	116	406
HTTP Server	1	1	2	3	5	10	22	35	47	82	137
KVM	1	1	2	2	5	9	14	26	40	101	115
MPlayer	1	1	1	2	4	7	14	23	33	72	522
OpenLDAP	1	1	1	1	2	4	11	24	34	85	259
PHP	1	1	1	1	2	5	13	33	55	117	1508
Postgresql	1	1	1	1	4	9	21	42	68	149	992
Python	1	1	1	2	2	4	20	55	99	176	387
Subversion	1	1	1	1	3	7	19	41	63	138	266
VLC	1	1	1	1	3	7	12	25	38	73	524
Linguagem: C++											
Chrome	1,0	1,0	1,0	1,0	3,0	5,0	10,0	17,0	26,0	52,8	205,0
Firefox	1,0	1,0	1,0	1,0	2,0	5,0	10,0	22,0	37,0	86,0	1558,0
Inkscape	1	1	1	1	2	5	10	21	31	55	249
Media Player	1	1	1	1	3	7	15	33	54	141	2943
Mysql	1	1	1	1	3	5	11	23	38	103	895
OpenOffice	1	1	1	1	2	3	7	13	20	45	252
Linguagem: Java											
Eclipse	1,0	1,0	1,0	1,0	2,0	5,0	9,0	18,0	26,0	49,7	606,0
Open JDK8	1,0	1,0	1,0	1,0	2,0	3,0	8,0	17,0	27,0	60,0	596,0
Ant	1	1	1	1	2	5	11	20	29	54	126
Checkstyle	1	1	1	1	1	3	6	11	15	26	55
Eclipse Metrics	1	1	1	1	2	4	6	12	15	22	27
Findbugs	1	1	1	1	2	3	7	15	22	48	198
GWT	1	1	1	1	2	4	9	20	29	53	385
Hudson	1	1	1	1	2	3	6	12	19	43	149
JBoss	1	1	1	1	2	3	6	11	17	40	382
Kalibro	1	1	1	1	3	5	8	12	16	26	33
Log4J	1	1	1	1	2	4	8	15	23	53	123
Netbeans	1	1	1	1	2	4	9	16	23	46	1002
Spring	1	1	1	1	2	3	7	14	21	46	122
Tomcat	1	1	1	1	2	4	10	21	35	80	300

Tabela 4 – Percentis para métrica NOM extraídos de [Meirelles \(2013\)](#)

Meirelles (2013) observou, a partir dos resultados obtidos para cada métrica de código-fonte, como os mostrados na Tabela 4, que, para uma determinada linguagem de programação, é possível definir uma referência, a partir de um determinado percentil, e observar o conjunto de valores que são frequentemente observados. Por exemplo, na Tabela 4, considerando **Open JDK8** como uma referência e que as informações para esta métrica são representativas apenas a partir do 75º percentil, é possível observar o intervalo de 0 a 8, como **muito frequente**, o intervalo de 9 a 17 como **frequente**, 17 a 27 como **pouco frequente** e acima de 27 como **não frequente** (MEIRELLES, 2013).

Considerando que o trabalho de Meirelles (2013), analisou softwares livres com grande utilização, que são mostrados na Tabela 4, é possível utilizar os intervalos de frequência obtidos como uma evidência **empírica** de qualidade do código-fonte. Sendo assim, os intervalos de frequência obtidos por Meirelles (2013) foram renomeados tais como a Tabela 5, a fim de facilitar a interpretação de métricas de código-fonte (objetivo específico OE2).

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

Tabela 5 – Nome dos Intervalos de Frequência

Após a renomeação dos intervalos de frequência em intervalos qualitativos, observou-se que alguns softwares apresentam menores valores para métricas que outros. Na tentativa de considerar dois cenários, considerou-se dois softwares como referências para cada uma das métricas na linguagem de programação Java. Em um primeiro cenário, foi analisado o software que contia os menores valores percentis para as métricas. Já em um segundo cenário, foi considerado os valores percentis mais altos. Para o primeiro cenário foi considerado os valores do **Open JDK8**, já para o segundo foi considerado os valores do **Tomcat** como referência.

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

Tabela 6 – Configurações para os Intervalos das Métricas para Java

2.3.4 Código Limpo

Segundo [Martin \(2008\)](#), o código deve ser uma composição de instruções e abstrações que possam ser facilmente entendidas, uma vez que gasta-se a maior parte do tempo lendo-o para incluir funcionalidades e corrigir falhas. Dessa forma, ao longo dos anos foram desenvolvidas práticas e técnicas a fim de se gerar o "código limpo" o qual se tem facilidade de entendimento e manutenibilidade. Na Tabela 7 são apresentados alguns conceitos de limpeza de código propostos por [Martin \(2008\)](#) e [Beck \(2007\)](#) compilados por [Machini et al. \(2010\)](#).

Conceito de Limpeza	Descrição	Consequências de Aplicação
Composição de Métodos	Compor os métodos em chamadas para outros rigorosamente no mesmo nível de abstração abaixo.	<ul style="list-style-type: none"> • 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).	<ul style="list-style-type: none"> • 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.	<ul style="list-style-type: none"> • 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.	<ul style="list-style-type: none"> • 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.	<ul style="list-style-type: none"> • Menos Parâmetros passados pela Classe • Possível diminuição na coesão
Uso Excessivo de Herança	Localizar uso excessivo de herança e transformá-lo em agregação simples.	<ul style="list-style-type: none"> • 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.	<ul style="list-style-type: none"> • Maior Encapsulamento de Parâmetros

Tabela 7 – Conceitos de Limpeza extraídos de [Machini et al. \(2010\)](#)

Os trabalhos de [Marinescu e Ratiu \(2004\)](#), [Marinescu \(2005\)](#), [Moha, Guéhéneuc e Leduc \(2006\)](#), [Moha et al. \(2008\)](#), [Moha et al. \(2010\)](#) e [Rao e Reddy \(2007\)](#) mostraram que é possível detectar pedaços de código-fonte que podem ser melhorados com a utilização de métricas de código-fonte. Embasado em alguns destes trabalhos, [Machini et al. \(2010\)](#) construiu um mapeamento entre as métricas de código-fonte e as técnicas e práticas propostas por [Martin \(2008\)](#) e [Beck \(2007\)](#).

Neste mapeamento, cada conjunto correlacionado de métricas constitui um cenário de limpeza, onde é mostrado a correlação de um "conceito de limpeza", apresentado na Tabela 7, com métricas de código-fonte, apresentadas na Seção 2.3. No trabalho de [Machini et al. \(2010\)](#), a detecção é feita a partir de valores altos para as métricas de código-fonte. A fim de medir efetivamente a aplicação destes, considerou-se os valores altos como os valores obtidos pelos intervalos Regular e Ruim para a configuração **Open JDK8** tal como mostrado na Tabela 6.

Considerando inicialmente os cenários **Classe Pouco Coesa** e **Interface dos Métodos** extraídos de [Machini et al. \(2010\)](#), foram elaborados mais alguns cenários tal como se mostra na Tabela 8. Além disso, realizou-se um mapeamento inicial e não formal de padrões de projeto de [Gamma et al. \(1994\)](#) que poderiam resolver os problemas detectados pelos cenários de limpeza.

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 subdivisão da Classe	Intervalos Regulares e Ruins de LCOM4, RFC.	<i>Chain of Responsibilities, Mediator, Decorator.</i>
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.	<i>Facade, Template Method, Strategy, Command, Mediator, Bridge.</i>
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.	<i>Composite, Prototype, Decorator, Adapter.</i>
Classe com muitos 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 Cíclica e Quebrar os métodos.	Intervalos Regulares e Ruins de AMLOC, ACCM.	<i>Chain of Responsibilities, Mediator, Flyweight.</i>
Classe com muita Exposição	Parâmetros Privados	Grande Número de Parâmetros Públicos	Reduzir o Número de Parâmetros Públicos.	Intervalos Regulares e Ruins de NPA.	<i>Facade, Singleton.</i>
Complexidade Estrutural	Maximização da Coesão	Grande Acoplamento entre Objetos	Reduzir a quantidade de responsabilidades dos Métodos.	Intervalos Regulares e Ruins de CBO e LCOM4.	<i>Chain of Responsibilities, Mediator, Strategy.</i>

Tabela 8 – Cenários de Limpeza

3 *Data Warehousing*

Os principais fatores para a adoção de um programa de métricas em organizações de desenvolvimento de software são regularidade da coleta de dados; a utilização de uma metodologia eficiente e transparente nessa coleta; o uso de ferramentas (não-intrusivas) para automatizar a coleta; o uso de mecanismos de comunicação de resultados adequados para todos os envolvidos; o uso de sofisticadas técnicas de análise de dados; (GOPAL; MUKHOPADHYAY; KRISHNAN, 2005 apud SILVEIRA; BECKER; RUIZ, 2010).

Data Warehousing é uma coleção de tecnologias de suporte à decisão disposta a capacitar os responsáveis por tomar decisões a fazê-las de forma mais rápida (CHAUDHURI; DAYAL, 1997 apud ROCHA, 2000). Em outras palavras, trata-se de um processo para montar e gerenciar dados vindos de várias fontes, com o objetivo de prover uma visão analítica de parte ou do todo do negócio (GARDNER, 1998). Desta forma, é possível em um ambiente de *data warehousing* que as métricas de código-fonte sejam coletadas de fontes diversas em uma periodicidade definida, de forma automatizada e não intrusiva ao trabalho da equipe de desenvolvimento e que estas possam trazer a visibilidade da qualidade do código-fonte produzido pela equipe de desenvolvimento durante um determinado período de tempo (dias, meses, anos).

A Figura 3 descreve uma arquitetura geral de um ambiente de *Data Warehousing*, de tal forma que setas 1 e 2 representam o processo de *Extraction-Transformation-Load*; A seta 3 representa as consultas *On-Line Analytical Processing (OLAP)*; Por fim, a seta 4 representa a visualização dos dados; Cada um dos componentes da Figura 3 é descrito nas seções subsequentes.

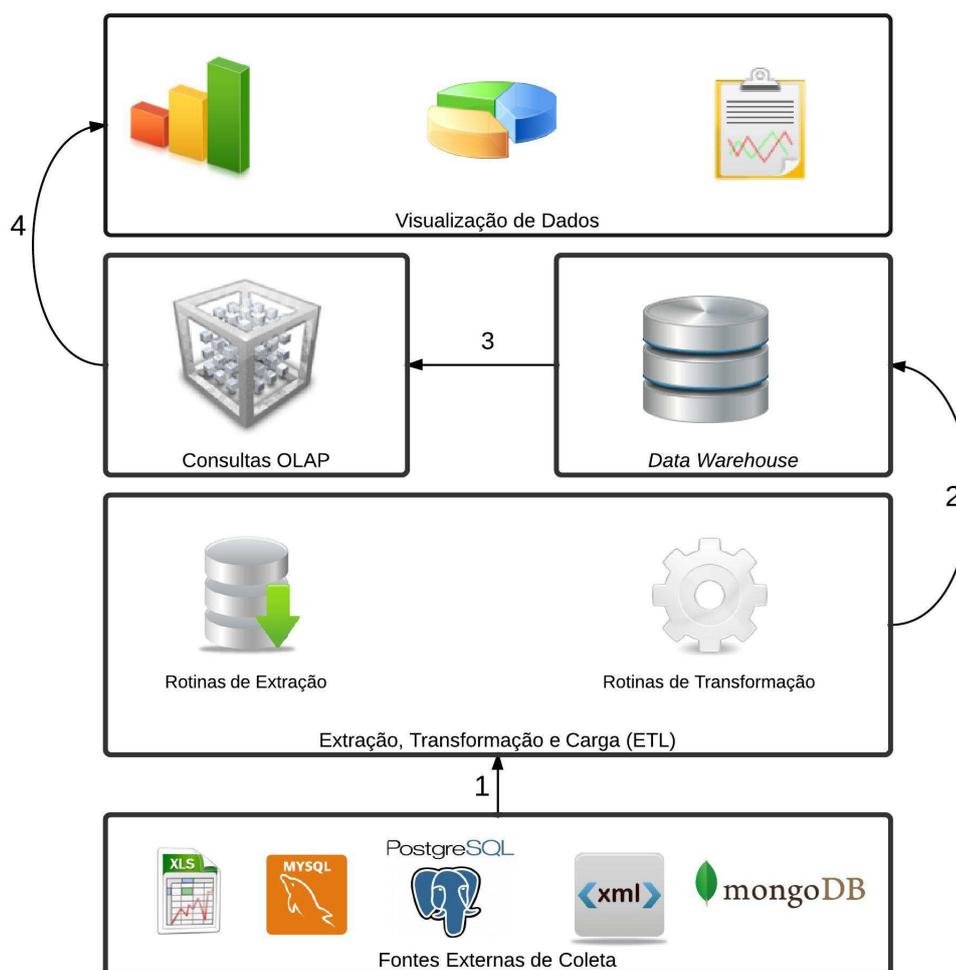


Figura 3 – Arquitetura de um ambiente de *Data Warehousing*

3.1 *Extraction-Transformation-Load* (ETL)

As etapas de extração, transformação, carga e atualização do *data warehouse* formam o back-end e caracterizam o processo chamado *Extraction- Transform-Load* (ETL). Esse processo pode ser dividido em três etapas distintas que somadas podem consumir até 85% de todo o esforço em um ambiente de *Data Warehousing* (KIMBALL; ROSS, 2002).

- **Extração:** No ambiente de *Data Warehousing*, os dados, que provêm de fontes distintas, tais como planilhas, bases relacionais em diferentes tipos de banco de dados (MySQL, Oracle, PostgreSQL e etc) ou mesmo de web services, são inicialmente extraídos de fontes externas de dados para um ambiente de *staging* que Kimball e Ross (2002) considera como uma área de armazenamento intermediária entre fontes e o *data warehouse*. Normalmente, o ambiente de *staging* é de natureza temporária e o seu conteúdo é apagado após a carga dos dados no *data Warehouse*.

- Transformação: Após os dados serem carregados na área de *staging*, os dados passam por processos de transformações diversas. Estas podem envolver desde uma simples transformação de ponto para vírgula até a realização de cálculos, como por exemplo, cálculos estatísticos.
- Carga: Após as devidas transformações dos dados, os dados são carregados, em formato pré-definido pelo projeto do *data warehouse*, em definitivo a fim de serem utilizados pelas consultas OLAP.

3.2 Data Warehouse

Data Warehouse é um conjunto de dados integrados, consolidados, históricos, segmentados por assunto, não-voláteis, variáveis em relação ao tempo e de apoio às decisões gerenciais (INMON, 1992). Ou seja, trata-se de um repositório central e consolidado que se soma ao conjunto de tecnologias que compõem um ambiente maior, que é o *Data Warehousing* (KIMBALL; ROSS, 2002).

A necessidade de centralização e agregação dos dados em um *data warehouse* mostrou que a modelagem relacional com a utilização das técnicas de normalização, que visam a eliminação da redundância de dados, não é eficiente quando se realiza consultas mais complexas que fazem uso frequente da operação JOIN entre várias tabelas, pois oneram recursos hardware com grandes quantidades de acesso físico a dados (KIMBALL; ROSS, 2002).

Dado esse cenário, Kimball e Ross (2002) propôs que o *data warehouse* deve ser projetado de acordo com as técnicas de modelagem dimensional, que visam exibir os dados em níveis adequados de detalhes e otimizar consultas complexas (TIMES, 2012).

No modelo dimensional, são aceitos que as tabelas possuam redundância e esparsidade de dados e estas podem ser classificadas em tabelas fatos e tabelas dimensões. Estas contêm dados textuais, que pode conter vários atributos descritivos que expressem relações hierarquizáveis do negócio. Já uma tabela fato é uma tabela primária no modelo dimensional onde os valores numéricos ou medidas do negócio são armazenados (KIMBALL; ROSS, 2002).

Quando se juntam fatos e dimensões, obtém-se o chamado esquema estrela, tal como se mostra na Figura 4. No modelo dimensional, ao se fazer uso da terceira forma normal entre as tabelas dimensões, o modelo passa então a ser chamado por modelo em *snowflake*, cujo ganho de espaço é menor que 1% do total necessário para armazenar o esquema do *data warehouse* (TIMES, 2012 apud KIMBALL; ROSS, 2002). Em ambos os casos, quando se relaciona três dimensões, obtém-se os cubos de dados (KIMBALL; ROSS, 2002), tal como se mostra na Figura 5.

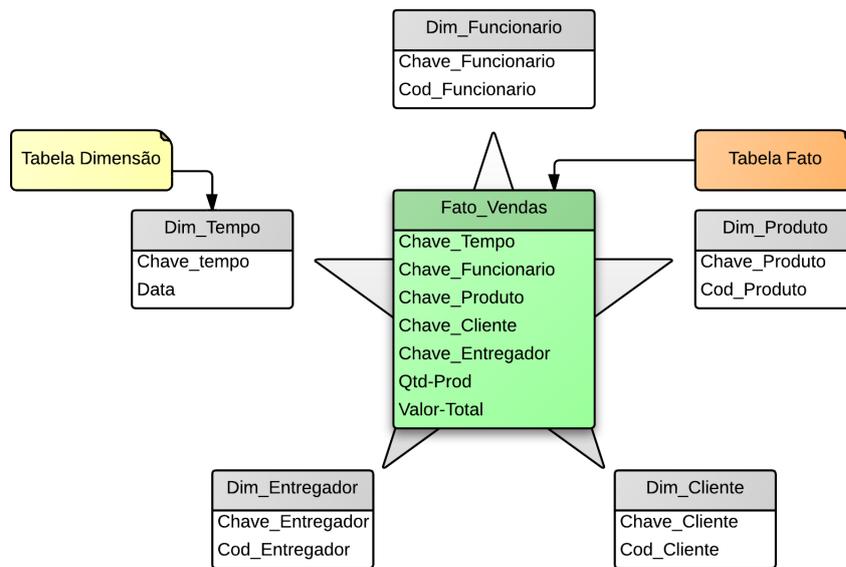


Figura 4 – Exemplo de Esquema Estrela adaptado de [Times \(2012\)](#)

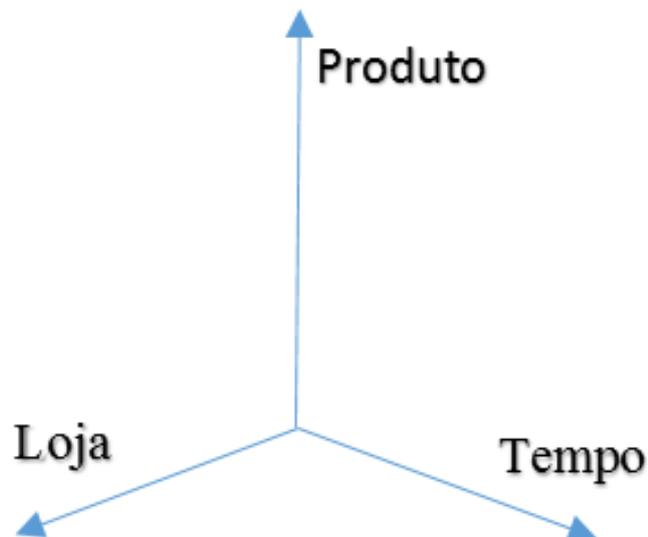


Figura 5 – Exemplo de Cubo de Dados

No esquema da Figura 4, percebe-se que uma tabela fato expressa um relacionamento muitos para muitos com as tabelas dimensões, mostrando assim que a navegabilidade dos dados quantitativos e qualitativos é mais intuitiva quando comparada com o modelo relacional normalizado até a terceira forma normal ([KIMBALL; ROSS, 2002](#)). Além disso, verifica-se que a tabela fato possui uma dimensão temporal associada, isto é, há fatos que ocorrem diariamente, como, por exemplo, a venda de produtos em um supermercado.

É possível que as vendas sejam vistas por visões mensais, trimestrais, semestrais ou anuais. Logo, a granularidade dos fatos deve ser considerada na hora de projetar um *data warehouse*. Além disto, deve-se considerar ainda as características do fato, pois quando os registros de uma tabela fato podem ser somados a qualquer dimensão, é dito que o fato é aditivo. Quando é possível apenas somar em relação a algumas dimensões, é dito que o fato é semiaditivo. Já quando o fato é usado apenas para registro e não pode ser somado em relação a nenhuma dimensão, é dito que o fato é não aditivo (INMON, 1992).

3.2.1 Metodologia do Projeto do *Data Warehouse*

A Figura 6 mostra os passos necessários, utilizando a metodologia proposta por Kimball e Ross (2002), para se projetar um *Data Warehouse*.



Figura 6 – Metodologia de Projeto de *Data Warehouse* proposta por Kimball e Ross (2002)

O primeiro passo de selecionar o processo de negócio é crucial na identificação de requisitos e regras de negócio, pois dele advém informações tais como cálculos específicos que podem ser entradas para o processo de *Extraction-Transformation-Load*. No segundo passo, 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. O terceiro e quarto passos resultam por fim no modelo dimensional que é apresentado na seção 3.2.

Após os fatos e dimensões serem identificados, necessita-se construir o projeto conceitual do *data warehouse*. Embora seja aplicada na construção de modelos relacionais, a construção de um diagrama de entidade relacionamento não pode ser aplicado em modelos dimensionais, pois não ficam expressas as navegabilidades entre as hierarquias (GOLFARELLI; MAIO; RIZZI, 1998).

Visando a construção de um modelo para representar um projeto conceitual de *data warehouse*, Golfarelli, Maio e Rizzi (1998) propôs o modelo dimensional-factual ou também conhecido como diagrama de árvore. Neste modelo, as dimensões são as raízes das árvores, se forem atributos não numéricos; as hierarquias são os “galhos” das árvores cujas raízes são dimensões e cujos relacionamentos entre os nós são 1:N; os atributos do fato, também são raízes das árvores, contudo estes são atributos numéricos caracterizados pela ausência de hierarquias.

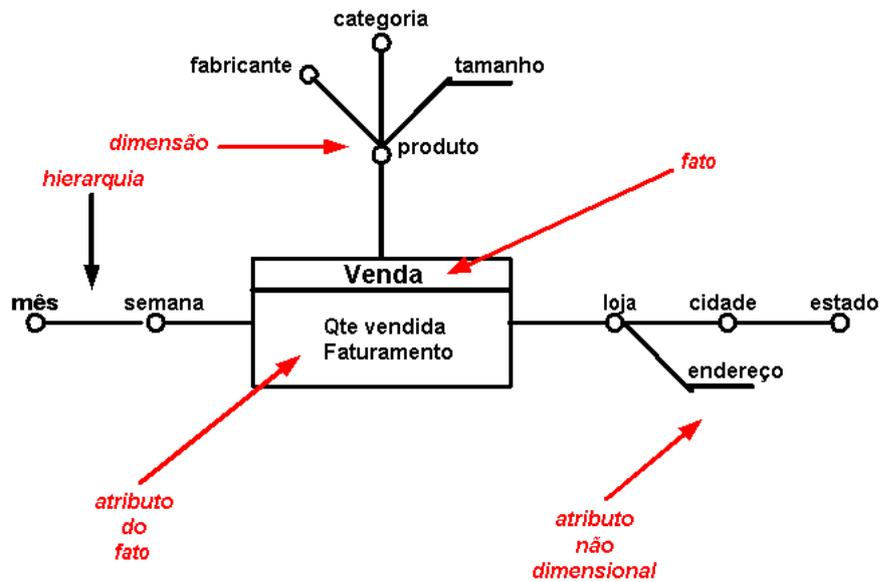


Figura 7 – Diagrama Dimensional de Árvore adaptado de [Golfarelli, Maio e Rizzi \(1998\)](#) e [Sampaio \(2007\)](#)

3.3 *On-Line Analytical Processing* (OLAP)

O termo OLAP, inicialmente proposto por [Codd, Codd e Salley \(1993\)](#), é utilizado para caracterizar as operações de consulta e análise em um *data warehouse* projetado sobre um modelo dimensional ([KIMBALL; ROSS, 2002](#)). Isto permite consultas mais flexíveis quando comparadas com as consultas *Online Transaction Processing* (OTLP) que são executadas em bancos de dados relacionais, frequentemente normalizados até a quinta forma normal, em que se visa a eliminação da redundância de dados.

As principais diferenças das operações *On-Line Analytical Processing* (OLAP) para as operações *Online Transaction Processing* (OTLP) são apresentadas na Tabela 9.

OLAP	OLTP
Modelagem Dimensional (Tabelas Fato e Dimensão).	Modelagem Relacional com a utilização das formas normais (3N, 4N, 5N).
Dados armazenados em nível transacional e agregado.	Dados em nível em nível transacional.
Visa o diminuir o uso do JOIN.	Faz uso constante de Join.
Análise de Dados.	Atualização de dados.
Estrutura de tipicamente estática.	Estrutura tipicamente dinâmica.
Proveem informações atuais e do passado.	Geralmente sem suporte a estado temporal dos dados.

Tabela 9 – Diferenças entre OLAP e OLTP extraídas de [Times \(2012\)](#), [Rocha \(2000\)](#) e [Neri \(2002\)](#)

Segundo [Neri \(2002\)](#), a consolidação é uma das mais importantes operações OLAP. Ela envolve a agregação de dados sobre uma ou mais hierarquias de dimensões. A generalização de uma consulta de consolidação pode ser representada formalmente através de:

```

Select P, F1(m1), ..., Fp(mp)
From C(D1(A11), ..., Dn(An+1))
Where φ(D1) and ... and φ(Dn)
Group by G

```

onde P representa os atributos a serem selecionados das dimensões. $F_i(m_i)$ para $(1 \leq i \leq p)$ representando uma função de agregação. A cláusula **From** $C(D_1(A_{11}), \dots, D_n(A_{n+1}))$ indica que a fonte de dados está indexada por suas tabelas dimensões, sendo que cada uma destas é referenciada como $D_i \dots D_n$ onde D_i contém K_i atributos de $D_i(A_{i1}), \dots, D_i(A_{ik_i})$ que descrevem a dimensão. A cláusula **Where** $\phi(D_i)$ é o predicado $(D_i(A_{ij}) = v_{ij})$, onde $v_{ij} \in \text{dom}(D_i(A_{ij}))$ onde $(1 \leq i \leq n)$ e $(1 \leq j \leq K_i)$. A cláusula **Group by** $G \subset D_i(A_{ij})$ tal que $(1 \leq i \leq n)$ e $(1 \leq j \leq K_i)$.

As operações OLAP tem como objetivo prover visualização dos dados sob diferentes perspectivas gerenciais e comportar todas as atividades de análise. Estas podem ser feitas de maneira *ad hoc*, por meio das ferramentas de suporte a operações OLAP. Contudo, há algumas que são documentadas pela literatura e são classificadas em dois grupos: Análise Prospectiva e Análise Seletiva ([CHAUDHURI; DAYAL, 1997](#) apud [ROCHA, 2000](#)).

A análise prospectiva consiste em realizar a análise a partir de um conjunto inicial

de dados para chegar a dados mais detalhados ou menos detalhados (INMON, 1992). Já a análise seletiva tem como objetivo selecionar pedaços do conjunto de dados para análise (ROCHA, 2000). Entre as operações de análise prospectiva estão:

- *Drill-Down*: Descer no nível de detalhes dos dados de uma dimensão. isto é, adicionar cabeçalhos de linha de tabelas de dimensão (KIMBALL; ROSS, 2002).
- *Roll-Up*: contrário de Drill-Down, trata-se caminhar para a visão de dados mais agregados (KIMBALL; ROSS, 2002 apud ROCHA, 2000).

Considerando o exemplo do total de vendas no mês de novembro em uma rede de lojas, que agregam as Lojas Sul, Norte e Oeste, tal como se mostra a Tabela 10, a operação Drill-Down pode ser exemplificada quando se adiciona a dimensão Produto na Tabela 10, isto é, aumentando o nível de detalhes, tendo então como resultado a Tabela 11. Já a operação de Roll-Up é o contrário, isto é, diminuir o nível de detalhe partindo da Tabela 11 para Tabela 10.

Mês	Loja	Total de Unidades Vendidas
Novembro	Loja Sul	200
Novembro	Loja Norte	300
Novembro	Loja Oeste	230

Tabela 10 – Exemplo do Total de Vendas de uma Rede de Lojas no mês de Novembro

Mês	Loja	Produto			
		Produto A	Produto B	Produto C	Produto D
Novembro	Loja Sul	10	70	50	70
Novembro	Loja Norte	100	60	50	90
Novembro	Loja Oeste	25	78	67	60

Tabela 11 – Exemplo do Total de Vendas de uma rede de lojas no mês de novembro com a dimensão Produto

- *Drill-Across*: significa caminhar a partir de uma dimensão para outra dimensão, combinando-as para mudar o enfoque da análise (ROCHA, 2000). O Drill Across pode ser aplicado à Tabela 10, obtendo assim a Tabela 12.

Loja Norte	
Produto	Novembro
Produto A	100
Produto B	60
Produto C	50
Produto D	60

Tabela 12 – Exemplo do Total de vendas da Loja Norte no mês de novembro

Entre as operações de análise seletiva estão:

- *Slice and Dice*: Em português, significa cortar e fatiar. Esta operação seleciona pedaços transversais do modelo dimensional e em seguida aplica critérios de seleção sobre este pedaço. (ROCHA, 2000). Ou seja, trata-se de uma operação semelhante a cláusula WHERE do SQL (TIMES, 2012). A operação pode ser aplicada na Tabela 13, obtendo assim a Tabela 14.

Produto	Loja	Outubro	Novembro	Dezembro
Produto A	Loja Sul	50	10	20
	Loja Norte	60	100	24
	Loja Oeste	70	25	53
Produto B	Loja Sul	32	70	20
	Loja Norte	42	60	43
	Loja Oeste	56	78	56
Produto C	Loja Sul	34	50	23
	Loja Norte	45	50	74
	Loja Oeste	83	67	65
Produto D	Loja Sul	56	70	35
	Loja Norte	12	90	34
	Loja Oeste	64	60	23

Tabela 13 – Exemplo de Vendas por produto de uma rede de lojas nos meses de novembro e dezembro

Produto	Loja	Outubro	Novembro
Produto A	Loja Sul	50	10
	Loja Norte	60	100
	Loja Oeste	70	25

Tabela 14 – Exemplo de Vendas do Produto A na rede de Lojas

- *Pivoting*: Trata-se de uma operação de rotação de 90° em um cubo multidimensional, isto é, muda-se a orientação das tabelas dimensionais a fim de se restringir a visualização das dimensões em uma tabela (ROCHA, 2000). A operação de Pivoting pode ser exemplificada ao partir da Tabela 13 para Tabela 15.

Loja	Produto	Outubro	Novembro	Dezembro
Loja Sul	Produto A	50	10	20
	Produto B	32	70	20
	Produto C	34	50	23
	Produto D	56	70	35
Loja Norte	Produto A	60	100	24
	Produto B	42	60	43
	Produto C	45	50	74
	Produto D	12	90	34
Loja Oeste	Produto A	70	25	53
	Produto B	56	78	56
	Produto C	83	67	65
	Produto D	64	60	23

Tabela 15 – Exemplo de Vendas por Loja para cada um dos Produtos nos meses de Novembro e Dezembro

3.4 Visualização de Dados

Dados transmitem importantes informações, logo cabe a quem deseja comunicá-los escolher a forma mais efetiva de fazê-lo (MINARDI, 2013). Segundo Minardi (2013), tabelas e gráficos são as formas mais comuns de transmitir as informações quantitativas, em que tabelas são utilizadas para consulta de valores individuais que podem ser comparados envolvendo, em certos casos, mais de uma unidade de medida; Já os gráficos são indicados para exibição de informação quantitativa nos quais os valores indicam pontos de interesse e estes podem ser comparados por suas similaridades e diferenças.

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

4.1 Arquitetura do Ambiente *Data Warehousing* para Métricas de Código-Fonte

Para a implementação do ambiente de *Data Warehousing* para métricas de código-fonte, foi definida a arquitetura tal como mostra a Figura 8.

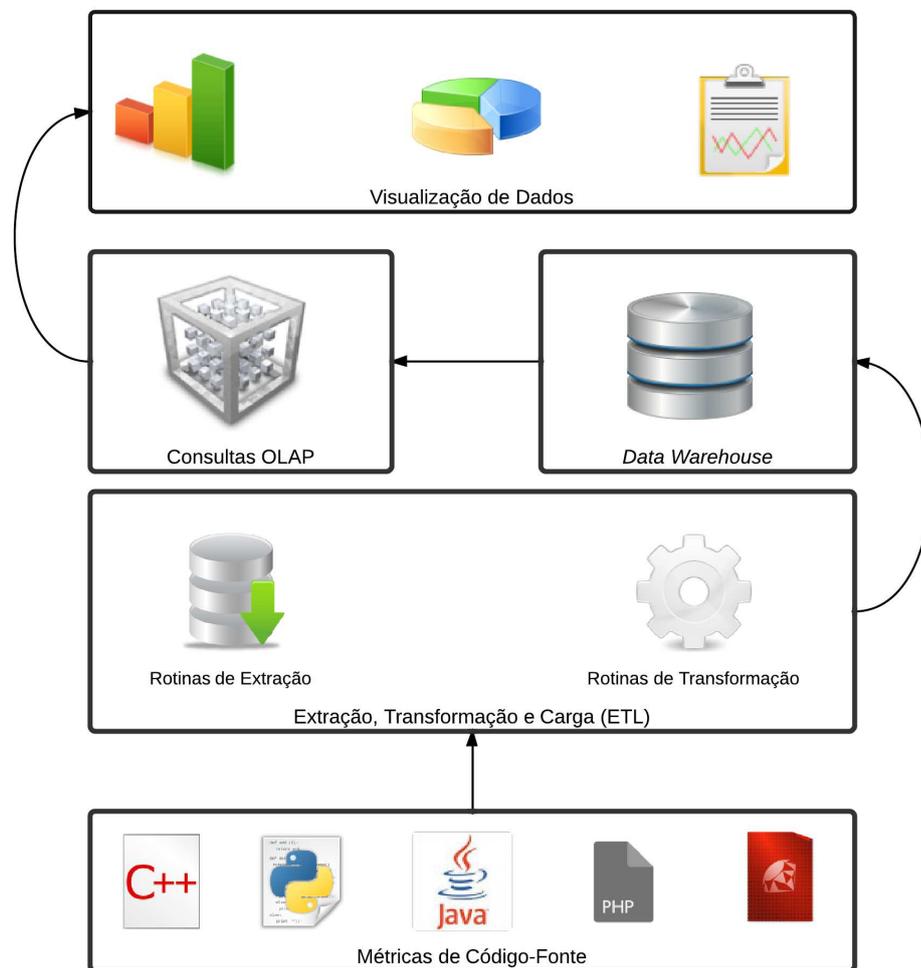


Figura 8 – Arquitetura do Ambiente de *Data Warehousing* para Métricas de Código-Fonte

Para selecionar as ferramentas, que implementariam cada um dos componentes, estabeleceram-se critérios gerais de seleção tal como pode ser visto na Tabela 16.

Identificador	Critério
CG01	A ferramenta deve possuir código aberto.
CG02	A ferramenta deve ter documentação disponível em inglês ou português.
CG03	A ferramenta deve possuir uma comunidade ativa em seu uso.
CG04	A ferramenta deve possuir releases estáveis.

Tabela 16 – Critérios Gerais de seleção de ferramentas

4.2 Ferramenta de Análise Estática de Código-Fonte

Além dos critérios gerais estabelecidos, para a escolha da ferramenta de análise estática de código-fonte, que é a fonte externa de coleta dos dados, estabeleceram-se os critérios específicos para seleção de ferramentas de análise estática de código fonte (CAE) apresentados na Tabela 17.

Identificador	Critério
CAE01	A ferramenta deve prover as métricas de código-fonte para as linguagens de programação, tal como especificado na Tabela 6.
CAE02	A ferramenta deve possuir saída de dados em arquivo em alguns dos seguintes formatos: JSON, XML, TXT, CSV.

Tabela 17 – Critérios Específicos para Ferramenta de Análise Estática de Código-Fonte

Após a realização de uma busca por ferramentas de análise estática de código-fonte, foram selecionados o SonarQube ¹ e o Analizo ² cujas principais características de ambas são apresentadas na Tabela 18.

¹ Disponível em <<http://www.sonarqube.org/>>

² Disponível em <<http://http://analizo.org/>>

Característica		
Linguagens com Suporte	C, C++, Java	C, C++, Java, PHP, Scala, Python, Delphi, Pascal, Flex, ActionScript, Javascript, Groovy ³ .
Licença	GNU GPL3.	GNU LGPL3.
Métricas de Código-Fonte fornecidas	25 métricas em âmbito de Projeto e 16 métricas em âmbito de Classe (MEIRELLES, 2013).	12 métricas em âmbito de Classe, 8 métricas em âmbito de projeto ⁴ .
Formato de Saída das Métricas	YAML, CSV.	JSON, XML.
Plataforma	GNU Linux (homologado para distribuições baseadas em Debian).	Windows, Linux, Mac OS X e Servidores de Aplicação Java.
Integração com outras ferramentas	Mezuro, Kalibro.	Jenkins, Hudson, Mantis, JIRA, Crowd e entre outros.
Sistema de Controle de Versões	Git	Git
Idioma com Suporte	Inglês.	Inglês, Português, Japonês, Italiano, Chinês, Francês, Grego e Espanhol.
Idioma da Documentação	Inglês.	Inglês.
Última Versão Estável em 10/05/2013	1.18.0	4.2

Tabela 18 – Características do SonarQube e do Analizo

Após levantar as características gerais de cada ferramenta, foram comparadas (SonarQube e Analizo) quanto aos critérios gerais e aos critérios específicos para ferramentas de análise estática, tal como se mostra na Tabela 19.

³ O SonarQube oferece suporte comercial a outras linguagens, contudo foram listadas apenas que tem suporte por meio de *plugins* de código-aberto

⁴ O SonarQube forneceu suporte as estas métricas até a versão 4

Critério		
CG01	✓	✓
CG02	✓	✓
CG03	✓	✓
CG04	✓	✓
CAE01	✓	✗
CAE02	✓	✓

Tabela 19 – Análise do SonarQube e do Analizo quanto aos critérios gerais e quanto aos critérios específicos de ferramentas de análise estática

Em uma fase inicial do trabalho, foi feita a análise entre o Analizo e o SonarQube, que resultou na decisão inicial de se utilizar o SonarQube. Contudo, desde a versão 4.1, o SonarQube retirou as métricas: **LCOM4**, **RFC**, **DIT**, **NOC** e **NOM**⁵. Dado que este fato impacta o principal objetivo do trabalho, foi tomada a decisão de migrar para a ferramenta Analizo. Esta evoluiu e ganhou a possibilidade de emitir saídas das métricas em CSV que detalham nome da classe e as respectivas métricas, atendendo assim ao critério CAE02. Adicionalmente, o Analizo permitiu ao trabalho incorporar a análise das métricas **ANPM**, **AMLOC**, **CBO**, **NPA**, que como foi observado na Seção 2.3.4, são cruciais na detecção de cenários de limpeza de código-fonte.

4.3 Projeto do *Data Warehouse*

O *Data Warehouse* como elemento central do ambiente de *Data Warehousing* deve ser o primeiro a ser projetado (KIMBALL; ROSS, 2002). Isso ocorre, pois o DW deve ser dirigido ao negócio. Logo, a modificação do DW impacta principalmente na carga dos dados, na etapa de extração, transformação e carga, requerendo modificações conforme o DW venha a mudar.

Seguindo a metodologia de Kimball e Ross (2002) para o projeto de *data warehouse*, apresentada na Seção 3.2.1, foi identificado o layout da fonte de dados como se mostra na Figura 9.

⁵ CoreMetrics do SonarQube: <<https://github.com/SonarSource/sonarqube/blob/master/sonar-plugin-api/src/main/java/org/sonar/api/measures/CoreMetrics.java>>

```
{
  "filename": "src/main/java/br/gov/iphan/sicg/apresentacao/controllers/BemCodigoExternoController.java",
  "module": "br::gov::iphan::sicg::apresentacao::controllers::BemCodigoExternoController",
  "acc": "0",
  "accm": "1",
  "amloc": "7.666666666666667",
  "an": "0",
  "anpm": "2",
  "asom": "0",
  "auv": "0",
  "bd": "0",
  "bf": "0",
  "cbo": "5",
  "da": "0",
  "dbz": "0",
  "df": "0",
  "dit": "0",
  "dnp": "0",
  "dupv": "0",
  "fgbo": "0",
  "lcom4": "6",
  "loc": "46",
  "mlk": "0",
  "mmloc": "15",
  "noa": "2",
  "noc": "0",
  "nom": "6",
  "npa": "0",
  "npm": "5",
  "obaa": "0",
  "osf": "0",
  "rfc": "26",
  "rsva": "0",
  "saigv": "0",
  "sc": "30",
  "ua": "0",
  "uaf": "0",
  "uav": "0"
},
```

Figura 9 – Fonte dos Dados do Processos de Negócio: *Layout* do Arquivo JSON.

Após a identificação da fonte de dados foram identificados, os processos de negócio e seus respectivos requisitos como se mostra nas Tabelas 20 e 21.

Requisito de Negócio	Descrição do Requisito de Negócio
RN1	Visualizar o intervalo qualitativo obtido para cada métrica de código-fonte em uma determinada release do projeto para a configuração Open JDK8 Metrics
RN2	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
RN3	Visualizar o o valor percentil obtida para cada métrica de código-fonte em uma determinada release do projeto para a configuração Open JDK8 Metrics
RN4	Comparar o o valor percentil a para cada métrica de código-fonte ao longo de todas as releases para a configuração Open JDK8 Metrics
RN5	Visualizar o intervalo qualitativo obtido para cada métrica de código-fonte em uma determinada release do projeto para a configuração Tomcat Metrics
RN6	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
RN7	Visualizar a medida obtida para cada métrica de código-fonte em uma determinada release do projeto para a configuração Tomcat Metrics
RN8	Comparar o valor percentil obtido para cada métrica de código-fonte ao longo de todas as releases para a configuração Tomcat Metrics

Tabela 20 – Requisitos de Negócio da Avaliação dos Valores Percentis das Métricas de Código-Fonte conforme as configurações especificadas na Tabela 6

Requisito de Negócio	Descrição do Requisito de Negócio
RN9	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 .
RN10	Comparar a quantidade de cenários de limpeza por tipo de cenários de limpeza de código-fonte em uma release de um projeto
RN11	Visualizar o total de cenários de limpeza em uma determinada release de um projeto
RN12	Visualizar cada uma das classes com um determinado cenário de limpeza de código-fonte ao longo das releases do projeto .
RN13	Visualizar as 10 classes de um projeto com menor número de cenários de limpeza identificados.
RN14	Visualizar as 10 classes de um projeto com maior número de cenários de limpeza identificados.
RN15	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 número total de classes da mesma release de um projeto

Tabela 21 – Requisitos de Negócio da Avaliação de Cenários de Limpeza de Código-Fonte e Avaliação de Taxa de Aproveitamento de Oportunidades de Melhoria de Código-Fonte conforme a Tabela 8

Aplicando o segundo passo da metodologia de [Kimball e Ross \(2002\)](#), identificou-se a menor granularidade dos dados como os dados mensais. Isso ocorre, pois mesmo que haja coletas diárias dos valores percentis das métricas de código-fonte e cenários de limpeza de código-fonte, a adição dos valores em relação a 30 coletas realizadas em um mês, por exemplo, não traz nenhuma informação representativa para o negócio. Ainda relativamente ao segundo passo da metodologia de [Kimball e Ross \(2002\)](#), identificou-se a periodicidade de coleta como as *releases* do software, isto é, cada software pode ter tempos diferenciados de lançamento de uma nova versão. Em alguns casos, estas podem ser semestrais ou mensais, contudo em outros casos, é possível obter *releases* diárias, como resultado da integração de um determinada massa de código em um sistema de integração contínua ([BECK, 1999a](#)).

Para aplicação do terceiro e quarto passo da metodologia de [Kimball e Ross \(2002\)](#), os fatos e as dimensões foram identificados a partir dos termos em negritos dos requisitos de negócio. Após a identificação dos fatos, estes foram classificados quanto a aditividade ⁶

⁶ [Kimball e Ross \(2002\)](#) enuncia que fatos que correspondem a porcentagens e taxas são considerados não aditivos e devem estar contidos, na mesma tabela, além do fato, o numerador e denominador. Este ocorre com a Taxa de Aproveitamento de Oportunidades de Melhoria de Código-Fonte

e relacionados com as respectivas dimensões conforme se mostra na Tabela 22.

Fato	Natureza do Fato	Dimensões
Valor Percentil	Não Aditivo	<ul style="list-style-type: none">• Projeto• Métrica• Configuração• Qualidade• <i>Release</i>• Tempo
Quantidade de Cenários de Limpeza	Aditivo	<ul style="list-style-type: none">• Projeto• Cenário de Limpeza• Classe• <i>Release</i>
Taxa de Aproveitamento de Oportunidade de Melhoria de Código-Fonte	Não Aditivo	<ul style="list-style-type: none">• Projeto• <i>Release</i>

Tabela 22 – Fatos e Dimensões do *Projeto de Data Warehouse*

Tendo os fatos identificados, foram criados os diagramas de árvore, tal como proposto por [Golfarelli, Maio e Rizzi \(1998\)](#), tal como se vê nas Figuras 10, 11 e 12.

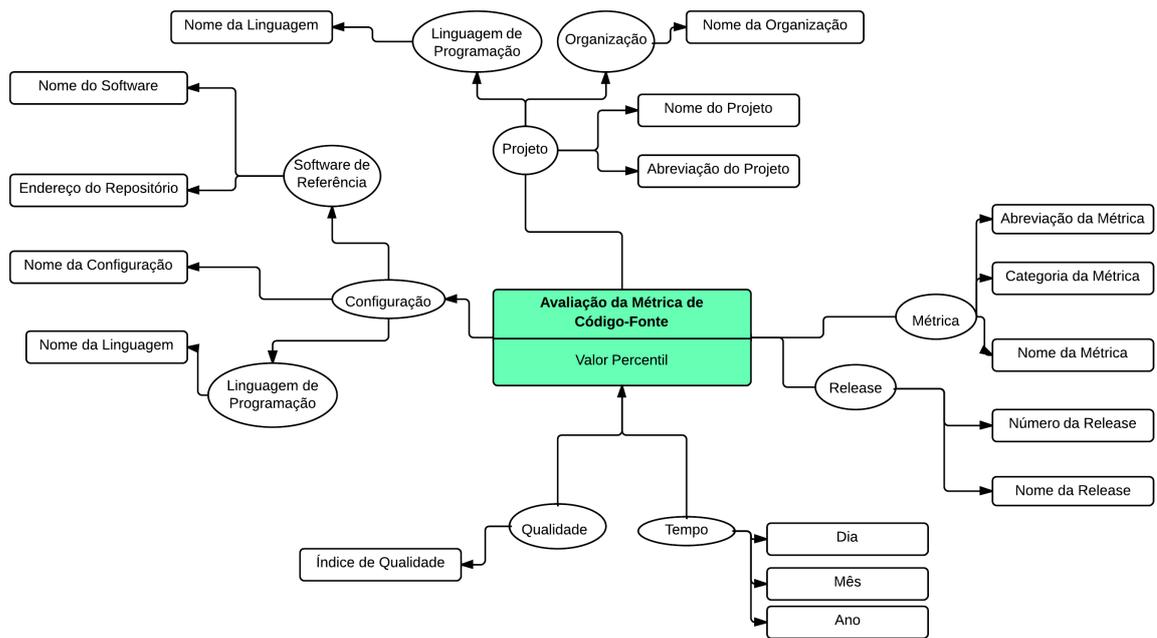


Figura 10 – Diagrama de Árvore para Valor Percentil das Métricas de Código-Fonte

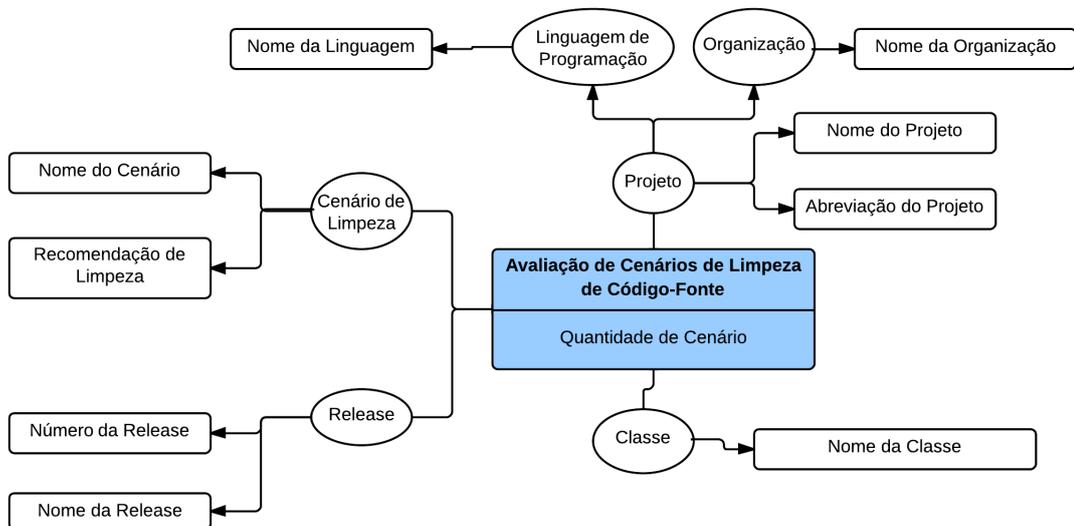


Figura 11 – Diagrama de Árvore para Avaliação de Cenários de Limpeza de Código-Fonte

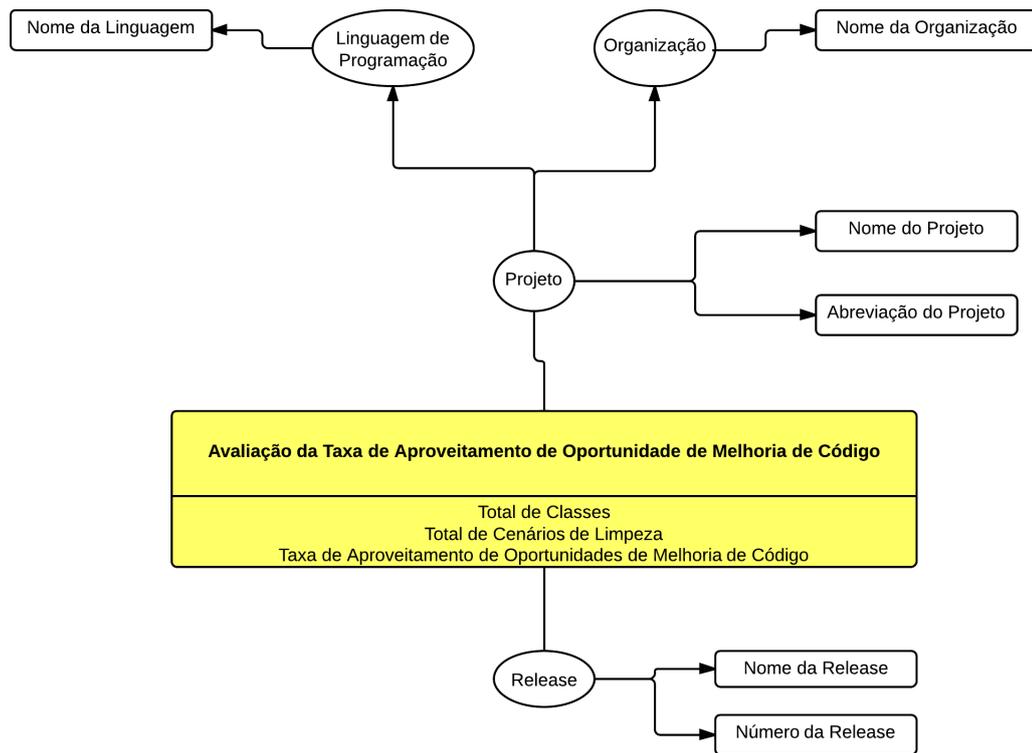


Figura 12 – Diagrama de Árvore para avaliação da Taxa de Aproveitamento de Oportunidade de Melhoria de Código-Fonte

Após a construção dos diagramas de árvore, o projeto físico do *Data Warehouse* foi construído utilizando a ferramenta MySQL Workbench. Como é possível observar na Figura 13, cada diagrama de árvore foi traduzido para cada tabela fato (tabelas com destaque verde) com as respectivas tabelas dimensões (tabelas com destaque azul).

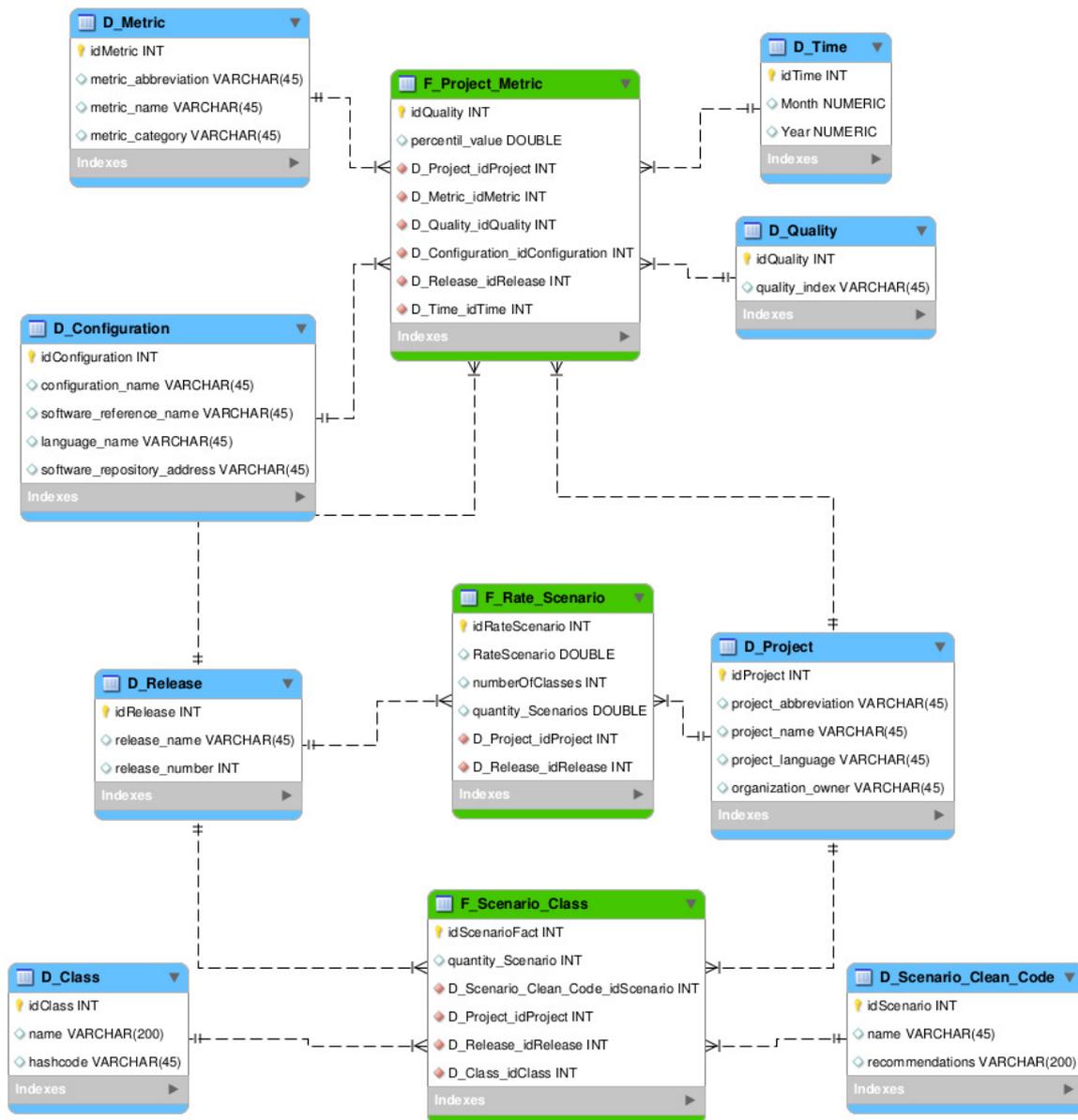
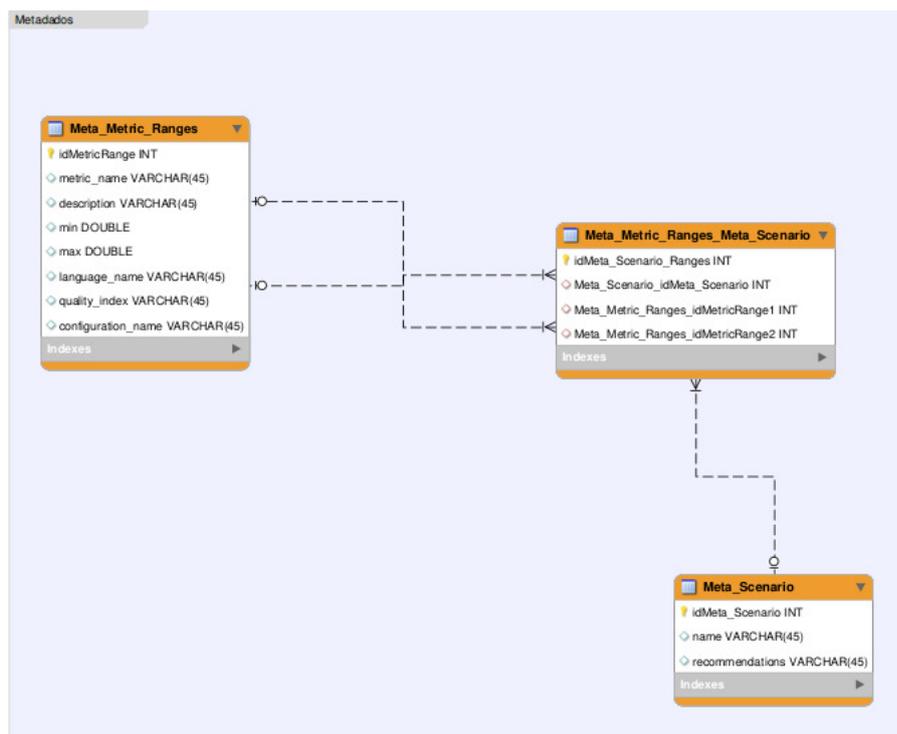


Figura 13 – Projeto Físico do *Data Warehouse*

Visando facilitar o processo de *Extraction-Transformation and Load*, decidiu-se criar uma área de metadados mostrada na Figura 14 com intuito de representar os dados sobre os dados dos processos de negócio.

Esta decisão de projeto foi motivada pela recomendação de que os metadados são todas as informações que constituem informações para os dados que farão parte do *Data Warehouse*, sendo que não há uma única forma de fazê-lo para atender as necessidades técnicas e administrativas. Além disso, a principal vantagem de utilização é facilitar a transformação de dados, no processo de ETL (KIMBALL; ROSS, 2002).

Figura 14 – Metadados do *Data Warehouse*

A Tabela "Meta_Metric_Ranges" contém cada uma das configurações de intervalos qualitativos para cada uma das métricas de código-fonte conforme descrito na Tabela 6. Já na Tabela "Meta_Scenario" estão contidos os cenários de limpeza de código-fonte, bem como as recomendações de limpeza para cada um dos cenários conforme descrito na Tabela 8.

Para atender o requisito de projeto, observou-se que cada cenário de limpeza identificado é composto por até dois intervalos qualitativos de métricas de código-fonte. Por este motivo, decidiu-se por criar a Tabela "Meta_Metric_Ranges_Meta_Scenario", com duas chaves estrangeiras da Tabela "Meta_Metric_Ranges", representando as associações entre os cenários de limpeza de código-fonte e os intervalos qualitativos de métricas de código-fonte.

4.4 Ferramentas de *Data Warehousing*

Tendo em vista que o *Data Warehouse* foi projetado em um modelo dimensional, é possível construir tanto o processo de *Extraction-Transformation-Load* quanto as operações de consulta OLAP. Entre as alternativas de código aberto que suportam este ambiente como um todo, está o Pentaho *Business Analytics Community Edition*. Esta solução de software livre apresenta soluções que cobrem as áreas de ETL, *reporting*, OLAP e mineração de dados. Cada um dos componentes que foi utilizado é apresentado e analisado nas seções subsequentes.

4.4.1 Implementação da Extração, Transformação e Carga dos Dados

O Pentaho *Data Integration Community Edition* ou Kettle⁷ é feito na linguagem Java e implementa o processo de ETL (Extração, Transformação e Carga de Dados). As principais características do Kettle bem como a análise quanto aos critérios gerais de seleção de ferramentas apresentadas na Tabela 23.

Característica		CG01	CG02	CG03	CG04
					
Licença	Apache License 2.0	✓			
Integração com Banco de Dados	MySQL, SQLServer, PostgreSQL, Oracle entre outros.				
Formatos Aceitos de Entrada de Dados	XML, TXT, JSON, ODS, XLS, CSV, Tabelas, YAML.				
Ultima Versão Estável (10/05/2014)	5				✓
Quantidade de Commits no Repositório Oficial	10.000			✓	
Idioma da Documentação	Inglês		✓		
Quantidade de Casos Abertos no <i>Issue Tracker</i>	2875			✓	

Tabela 23 – Características do Kettle e avaliação quanto aos critérios gerais de seleção de ferramentas

4.4.2 Implementação das Consultas OLAP e Visualização de Dados

Para a implementação das consultas OLAP e Visualização de dados, torna-se necessário a utilização do Pentaho BI Platform⁸, que é uma ferramenta que provê a arquitetura e a infraestrutura para soluções de *Business Intelligence*, *Data Mining* e a camada de visualização de dados do *Data Warehouse*.

O Pentaho BI Platform tem as principais características e a análise quanto aos critérios gerais de seleção de ferramentas apresentadas na Tabela 24.

⁷ Disponível em <<http://kettle.pentaho.com/>>

⁸ Disponível em <http://community.pentaho.com/projects/bi_platform/>

Característica		CG01	CG02	CG03	CG04
					
Licença	Apache License 2.0	✓			
Ultima Versão Estável (10/05/2014)	5				✓
Quantidade de Commits no Repositório Oficial (14/05/2014)	4000+			✓	
Idioma da Documentação	Inglês		✓		
Quantidade de Casos Abertos no <i>Issue Tracker</i> (14/05/2014)	2000+			✓	

Tabela 24 – Características do Pentaho BI Platform e avaliação quanto aos critérios gerais de seleção de ferramentas

A ferramenta Pentaho BI Platform possui arquitetura extensível por plugins diversos que realizam diversas operações, tais como, criação de relatórios, visualização dos dados em tabelas e gráficos, entre outros. Entre os plugins disponíveis, está o Saiku Analytics que oferece serviços de apoio a operações OLAP e à visualização de dados. As características gerais do Saiku Analytics, bem como a avaliação quanto aos critérios gerais de seleção de ferramentas, são apresentados na Tabela 25.

Característica		CG01	CG02	CG03	CG04
					
Licença	GPL 2.0	✓			
Componentes de Visualização	Gráfico de Pizza, Gráfico de Linhas, Gráfico de Área, Gráfico de Setor e entre outros.				
Ultima Versão Estável (10/05/2014)	2.8				✓
Idioma da Documentação	Inglês		✓		

Tabela 25 – Características do Saiku Analytics e avaliação quanto aos critérios gerais de seleção de ferramentas

Na arquitetura do Saiku Analytics, está incorporado outro software livre chamado de Mondrian OLAP. Por meio dele, é possível realizar consultas. Estas ocorrem por meio da escrita de *queries* em linguagem MDX (*Multidimensional eXpressions*), que foi proposta por Spofford et al. (2006) como uma forma de escrever consultas mais otimizadas para bases que seguem o modelo dimensional, tal como mostra o exemplo do trecho de Código-Fonte 1.

```
1  SELECT
2    { [Measures].[Loja] } ON COLUMNS ,
3    { [Tempo].[2002] , [Tempo].[2003]
4      } ON ROWS
5  FROM Vendas
6  WHERE ( [Loja].[Loja Sul])
```

Código-Fonte 1 – Exemplo de *Query* em
linguagem MDX

4.5 Resumo das Ferramentas utilizadas do Ambiente de *Data Warehousing*

Na Figura 15, é apresentado o modo como cada uma das ferramentas apresentadas na seções anteriores está disposta na arquitetura do ambiente de *Data Warehousing* para Métricas de Código-Fonte.

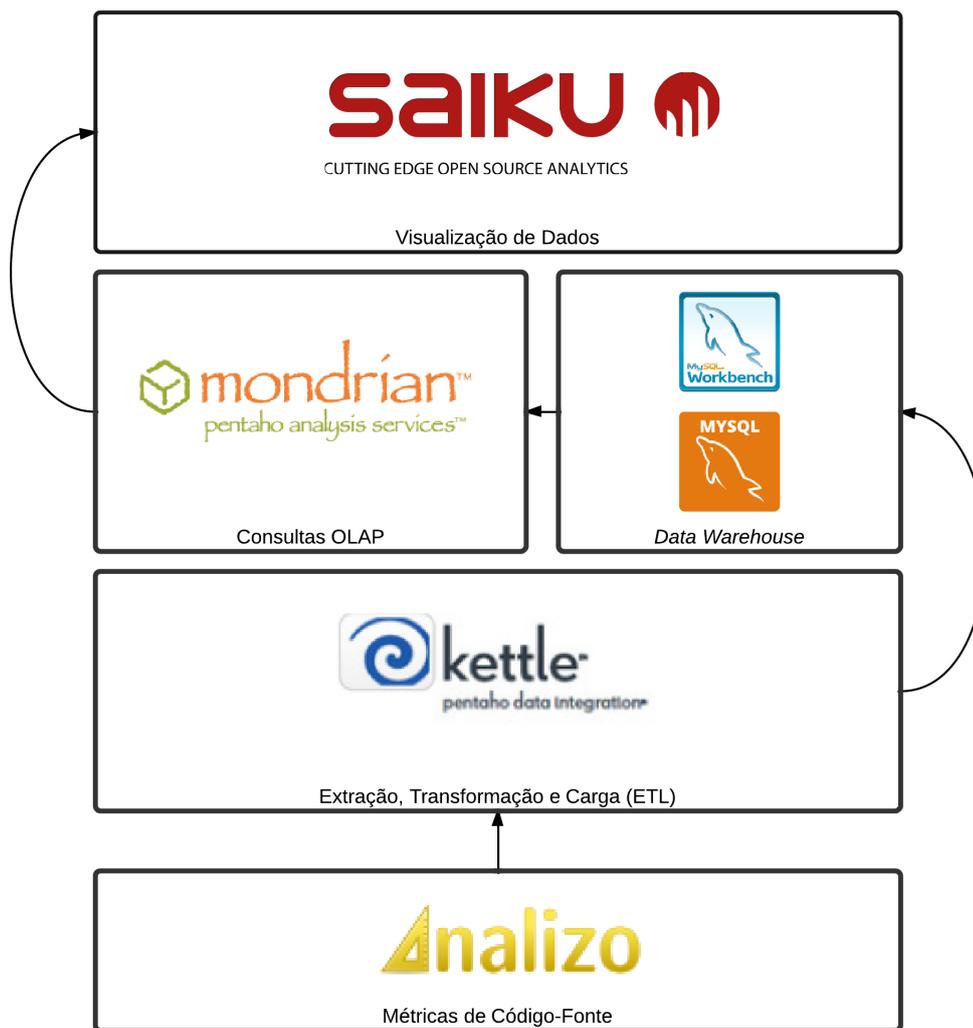


Figura 15 – Arquitetura Ambiente de *Data Warehousing* para Métricas de Código-Fonte

A arquitetura descrita na Figura 15 foi implementada em uma máquina virtual com as seguintes configurações:

- Processador: Intel(R) Xeon(R) CPU E5-2620 @ 2.00GHz
- RAM: 8GB
- Distribuição: Debian Whezzy

5 Estudo de Caso

Wohlin et al. (2012) enuncia a necessidade que uma metodologia de estudo de caso, definida formalmente em engenharia de software, deva conter os passos da Figura 16.

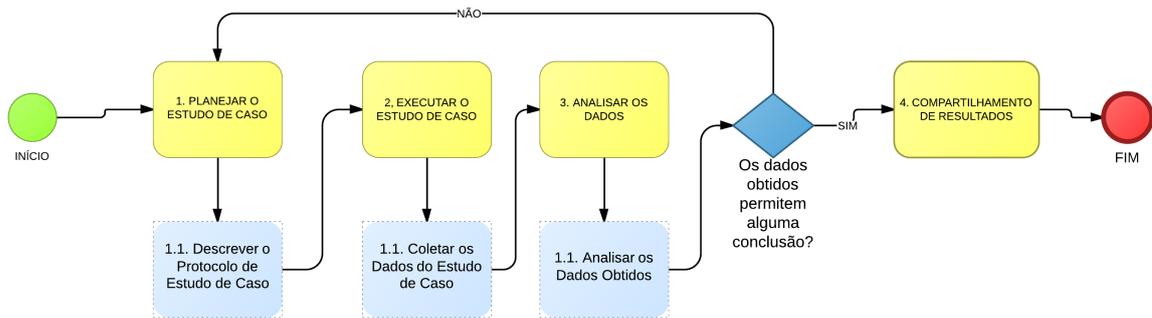


Figura 16 – Metodologia de Estudo de Caso proposta por Wohlin et al. (2012)

5.1 Planejamento do Estudo de Caso

Para o planejamento do estudo de caso em engenharia de software é essencial, segundo Brereton et al. (2008), que se descreva o protocolo do estudo de caso. Este consiste em um relatório simplificado das principais variáveis, dados e passos, tais como o meio e modo de coleta de dados, identificação das fontes de dados e formas de análise de dados (WOHLIN et al., 2012).

O protocolo de estudo de caso deste trabalho, que foi baseado em Brereton et al. (2008) e Wohlin et al. (2012) foi dividido em seções, conforme se vê a seguir.

5.1.1 Trabalhos Relacionados ao Estudo de Caso

Assim como os trabalhos de Palza, Fuhrman e Abran (2003), Ruiz et al. (2005), Castellanos et al. (2005), Becker et al. (2006), Folleco et al. (2007), Silveira, Becker e Ruiz (2010), buscou-se alcançar uma validação empírica da utilização do ambiente de *Data Warehousing* proposto no Capítulo 4.

5.1.2 Objetivos do Estudo de Caso

O objetivo geral do estudo de caso foi avaliar um software durante o seu desenvolvimento no ambiente de *Data Warehousing* proposto no Capítulo 4. Em consonância ao objetivo geral do estudo de caso e aos objetivos específicos deste trabalho, foram definidos objetivos específicos do estudo de caso, tal como se mostra na Tabela 26.

Identificador	Objetivo Específico do Estudo de Caso	Objetivo Específico do Trabalho
OEC1	Avaliar a qualidade do código-fonte em um projeto.	OE1
OEC2	Validar a automação do processo de medição de métricas de código-fonte no ambiente de <i>Data Warehousing</i> .	OE2
OEC3	Verificar a facilidade de interpretação das métricas de código-fonte.	OE3
OEC4	Acompanhar os indicadores de código-limpo no projeto.	OE4
OEC5	Avaliar a taxa de aproveitamento de melhoria de código-fonte.	OE5
OEC6	Quantificar as visualizações as consultas OLAP observadas no ambiente de <i>Data Warehousing</i> para se atender aos requisitos de negócio.	OE6

Tabela 26 – Objetivos Específicos de Estudo de Caso

5.1.3 Seleção de Objeto do Estudo de Caso

Para selecionar o software que seria parte da avaliação do ambiente de *Data Warehousing* que é o objeto do estudo de caso, foram considerados os critérios estabelecidos na Tabela 27.

Identificador	Critério
CEC1	Software desenvolvido na linguagem Java para que as métricas de código-fonte fossem avaliadas conforme as configurações definidas na Tabela 6.
CEC2	Lançamento de novas versões de forma iterativa e incremental a fim de observar o comportamento de dados ao longo do tempo.
CEC3	Disponibilidade do código-fonte para que o Análise pudesse extrair as métricas de código-fonte e estas pudessem ser posteriormente carregadas no ambiente de <i>Data Warehousing</i> como explicado no Capítulo 4.

Tabela 27 – Critérios de Seleção de Objeto do Estudo de Caso

5.1.4 Objeto de Estudo de Caso

O Sistema Integrado de Gestão do Conhecimento (SICG) do Instituto do Patrimônio Histórico e Artístico Nacional, a ser apresentado nas seções a seguir, foi escolhido como o software a ser analisado no ambiente de *Data Warehousing*. Este software também foi analisado, do ponto de vista contratual por outro estudo de caso proposto por Santos (2014) que analisou a influências da utilização do Scrum e o Pensamento Lean na gestão contratual do desenvolvimento deste software.

5.1.4.1 Visão Geral dos Requisitos

O IPHAN é autarquia federal responsável pela gestão de diversos processos de preservação do patrimônio cultural, como por exemplo, ações para sua identificação, proteção, gestão e fomento. Decorrente de suas atribuições, o órgão produz uma grande quantidade de informações fragmentadas em termos territoriais e temáticos. Nos últimos quatro anos, o IPHAN elaborou uma metodologia para definir os processos de cadastro, inventário e gestão do patrimônio cultural material. Essa metodologia tem por objetivo geral abordar o Patrimônio Cultural de forma integrada, sistêmica e estratégica, conforme detalhado a seguir:

- Integrada: cobrindo todas as categoriais do patrimônio material;
- Sistêmica: estabelecendo moldes a serem utilizados nas diversas etapas de ações de preservação, possibilitando o "diálogo" e troca de informações entre áreas e etapas de trabalho;
- Estratégica: considerando o mapeamento, a organização e a disponibilização de informações sobre o patrimônio como base para a construção de políticas públicas integradas - com outros parceiros - e de planos de preservação e desenvolvimento das regiões onde se inserem os bens.

Em termos específicos, a metodologia buscou, em primeiro lugar, mapear os procedimentos necessários para a execução das ações de cadastro, proteção, normatização e fiscalização de bens culturais de natureza material, indicando adicionalmente os dados a serem coletados. Este mapeamento contou com a participação de representantes das Superintendências e Escritórios Técnicos, que, por meio de Grupos de Trabalho, analisaram, de forma crítica, as metodologias até então existentes.

A revisão dos processos levou à formulação da nova metodologia que, por sua vez, permitiu a otimização das atividades de cadastro de sítios históricos e de bens tombados isoladamente e gerou a normalização das ações de fiscalização. O resultado desse trabalho produziu um conjunto de fichas e procedimentos específicos com demandas para cadastro de dados textuais, geográficos e imagens.

Há um entendimento no IPHAN de que é necessária a formação de uma rede de proteção fomentada pelo SNPC que consolide o grande volume de informações atualmente produzido por suas unidades administrativas, composto de 27 Superintendências, 30 escritórios técnicos, 4 Unidades Especiais e 2 Parques Históricos Nacionais. Entretanto, na conjuntura atual, a natureza das informações, em grande maioria armazenadas em planilhas e em banco de dados isolados, dificulta o processo de consolidação das informações, fato que impede a construção da rede de proteção baseada nos recursos e tecnologias atualmente adotados, pois demandaria o aporte considerável de recursos financeiros e humanos sem ganhos no processo. O órgão se manteria refém da demora na produção de informações decorrente do intervalo entre ação e recepção das respostas, ou seja, entre a percepção do problema e sua solução.

Apesar do fato de os processos da metodologia de cadastro, normatização de sítios urbanos tombados e fiscalização de bens imóveis já fazerem parte da realidade das Superintendências e Escritórios Técnicos, o processo manual de suporte e gestão dos dados torna a execução precária e morosa.

5.1.4.2 O Software Analisado

Tendo em vista a dificuldade que o processo manual acarreta, o IPHAN decidiu contratar uma empresa de software para desenvolver uma solução que pudesse automatizar o processo de trabalho decorrente da metodologia de inventário, cadastro, normatização, fiscalização, planejamento e análise e gestão do patrimônio material. Esta solução de software foi concebida sobre a denominação de Sistema Integrado de Conhecimento e Gestão (SICG) que foi construído em Java (atende-se ao critério CEC1), durante 24 *releases* mensais (atende-se ao critério CEC2), utilizando *frameworks* como VRaptor, Hibernate formado por 7 módulos:

- Módulo de Conhecimento;
- Módulo de Análise e Gestão;
- Módulo de Cadastro;
- Módulo de Administração de Usuários;
- Módulo de Fiscalização;
- Módulo de Cadastro Auxiliares;
- Módulo de Relatórios Adicionais.

5.1.5 Dados, Fonte dos Dados e Forma de Análise dos Dados

Após a identificação dos objetivos do estudo de caso, foram identificados as fontes de onde os dados serão retirados:

- Resultados da Análise Estática do Código-Fonte do SICG pelo Analizo
- Uso do Ambiente de *Data Warehousing*
- Entrevistas não estruturadas com o Gestor de Contrato do SICG
- Relatório de Auditoria sobre o Código-Fonte do SICG que foi realizado por uma empresa de TI.

Com a identificação das fontes de dados, pôde-se identificar os principais dados qualitativos e quantitativos na Tabela 28. Estes são as respostas que espera obter com o monitoramento e avaliação de métricas de código-fonte e cenários de limpeza de código-fonte em um ambiente de *Data Warehousing*.

Dado	Fonte do Dado	Forma de Análise do Dado
Valores Percentis de Métrica de Código-Fonte	Análise estática do Código-Fonte do Sistema Integrado de Conhecimento e Gestão (SICG) pelo Analizo.	Intervalos Qualitativos das Métricas de Código-Fonte conforme a Tabela 6.
Métricas de Código-Fonte no Relatório de Auditoria de Código-Fonte	Relatório de Auditoria de Código-Fonte	Comparação das análises realizadas pela empresa com as obtidas pelos Valores Percentis de Métrica de Código-Fonte.
Cenários de Limpeza de Código-Fonte	Interpretação dos Valores da Análise estática do Código-Fonte do Sistema Integrado de Conhecimento e Gestão (SICG) obtidos pela ferramenta Analizo para cada Classe.	Este Dado foi interpretado conforme a Tabela 8.
Taxa de Aproveitamento de Oportunidade de Melhoria de Código-Fonte	Cálculo realizado a partir divisão do número de cenários de limpeza de código-fonte identificados no código-fonte do Sistema Integrado de Conhecimento e Gestão (SICG) e número total de classes.	Se esta taxa apresenta valores mais altos com passar o tempo, significa dizer que o projeto não está aproveitando as oportunidades de melhoria de código-fonte.
Consultas OLAP realizadas no Ambiente de <i>Data Warehousing</i>	Número de consultas OLAP necessárias para se atender aos requisitos de negócio.	Para cada requisito de negócio, conta-se as diferentes consultas OLAP necessárias.

Tabela 28 – Dados a serem analisados no Estudo de Caso

5.1.6 Ameaças à Validade do Estudo de Caso

Yin (2011) destaca que há quatro ameaças a realização de estudos de caso: Validade de Construção, Validade Interna, Validade Externa e de Confiabilidade.

A validade de construção está presente na fase de coleta de dados onde 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, 2011). Neste trabalho, buscou-se garantir

a validade de construção ao se definir objetivos com evidências diferentes. Estas, por sua vez estão, diretamente relacionadas com os objetivos do estudo de caso e os objetivos do trabalho conforme mostrado na Tabela 26.

A validade interna é garantida quando as conclusões apresentadas pelo estudo de caso correspondem a alguma realidade conhecida pelos próprios envolvidos no projeto (YIN, 2011). Para se atender a validade interna, foi realizada a triangulação entre várias fontes de dados: os dados analisados pelo ambiente de *data warehousing*, entrevistas não estruturadas e relatório de auditoria de código-fonte.

Quanto a validade externa, destaca-se que a utilização de um estudo de caso não é suficiente para generalizar os resultados dele obtidos, sendo necessário a utilização de estudo em múltiplos casos, a fim de comprovar resultados genéricos (YIN, 2011). Embora, como citado anteriormente, alguns trabalhos já tenham mostrado a utilização de métricas e ambientes de *Data Warehousing*, não há como correlacionar os resultados obtidos devido a grandes diferenças de contexto e aplicação do ambiente, logo a validade externa não é garantida neste trabalho.

Com relação a confiabilidade, Yin (2011) enuncia que para garanti-la em um estudo de caso, é necessário que seja reproduzido caso seja usada a mesma fonte dos dados. Neste trabalho, com a documentação da implementação do ambiente de *Data Warehousing*, presente no Capítulo 4, conjuntamente com o protocolo de estudo de caso, apresentado a partir da Seção 5.1, garante-se a repetição do estudo de caso e por conseguinte a confiabilidade.

5.2 Execução do Estudo de Caso e Análise dos Dados

Para cada uma das *releases* do SICG, coletou-se o código-fonte, conforme a disponibilidade, em um servidor FTP cedido pelo IPHAN. Em alguns casos, o próprio IPHAN não possuía o código-fonte de uma determinada *release* tal como se mostra na Figura 17.

Informações do Repositório do Projeto IPHAN	
Sprint	Informações
Sprint 1	Perdido
Sprint 2	Adequada
Sprint 3	Perdido
Sprint 4	Adequada
Sprint 5	Adequada
Sprint 6	Perdido
Sprint 7	Adequada
Sprint 8	Adequada
Sprint 9	Fora do padrão
Sprint 10	Fora do padrão
Sprint 11	Fora do padrão
Sprint 12	Fora do padrão
Sprint 13	Adequada
Sprint 14	Fora do padrão
Sprint 15	Perdido
Sprint 16	Fora do padrão
Sprint 17	Adequada
Sprint 18	Perdido
Sprint 19	Fora do padrão
Sprint 20	Fora do padrão
Sprint 21	Fora do padrão
Sprint 22	Fora do padrão
Sprint 23	Fora do padrão
Sprint 24	Fora do padrão
Legenda	
	Código Perdido
	Código com estrutura de diretórios adequada
	Código fora do padrão de estrutura de diretórios

Figura 17 – Informações do Repositório de Código-Fonte

Após o código-fonte de cada versão ser coletado, foi feita uma análise da estrutura de diretórios a fim de obter uma padronização na análise das métricas de código-fonte. Em casos que foram marcados com amarelos conforme a Figura 17, a estrutura de diretórios não estava condizente com a adotada pelo projeto, sendo necessário a correção para que as análises de código-fonte pudessem ser executadas corretamente. Posteriormente, os dados obtidos da análise estática do código-fonte foram carregados no ambiente de *Data Warehousing* proposto no Capítulo 4. Dessa forma, foi possível obter para cada *release* que estava com código-fonte presente: Valor Percentil para cada uma das Métricas de Código-Fonte, Quantidade de Cenários de Limpeza presentes em uma determinada Classe do Projeto e o cálculo da Taxa de Aproveitamento de Oportunidade de Melhoria de Código-Fonte.

5.2.1 Análise dos Valores Percentis para Métricas de Código-Fonte do SICG

Após extrair as métricas do código-fonte, obteve-se valores percentis para as configurações "OpenJDK8 Metrics" e "Tomcat Metrics" , que foram definidas na Tabela 6, conforme mostra as Figuras 18 e 19 respectivamente.

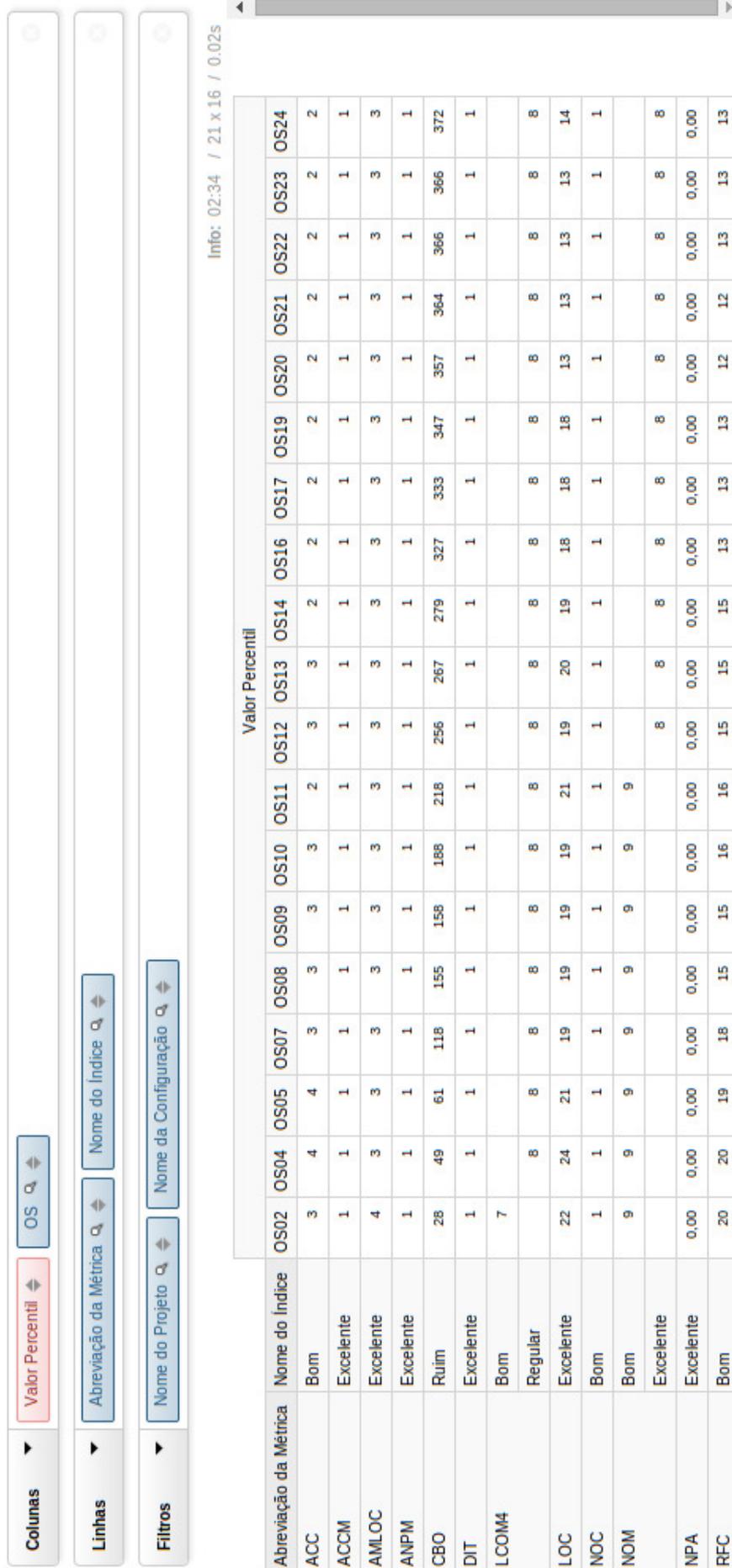


Figura 18 – Interpretação dos valores percentis conforme a configuração "OpenJDK8 Metrics"

Colunas ▾

Valor Percentil

Linhas ▾

Abreviação da Métrica

Filtros ▾

Nome do Projeto

Info: 02:32 / 21 x 15 / 0.02s

Abreviação da Métrica	Nome do Índice	Valor Percentil																			
		OS02	OS04	OS05	OS07	OS08	OS09	OS10	OS11	OS12	OS13	OS14	OS16	OS17	OS19	OS20	OS21	OS22	OS23	OS24	
ACC	Bom	3	4	4	3	3	3	3	3	2	3	2	2	2	2	2	2	2	2	2	
ACCM	Excelente	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
AMLOC	Excelente	4	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	
ANPM	Excelente	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
CBO	Ruim	28	49	61	118	155	158	188	218	256	267	279	327	333	347	357	364	366	366	372	
DIT	Excelente	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
LCOM4	Bom	7																			
	Regular		8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	
LOC	Excelente	22	24	21	19	19	19	19	21	19	20	19	18	18	18	13	13	13	13	14	
NOC	Bom	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
NOM	Excelente	9	9	9	9	9	9	9	9	8	8	8	8	8	8	8	8	8	8	8	
NPA	Excelente	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	
RFC	Bom	20	20	19	18	15	15	16	16	15	15	15	13	13	12	12	13	13	13	13	

Figura 19 – Interpretação dos valores percentis conforme a configuração "Tomcat Metrics"

As métricas **ACCM**, **AMLOC**, **ANPM**, **DIT**, **LOC** e **NPA** permaneceram no intervalo qualitativo "Excelente" tanto para a configuração "OpenJDK8 Metrics" quanto para a configuração "Tomcat Metrics", mostrando assim que, a partir da comparação do SICG com estes software de referência, pode-se inferir um bom nível de qualidade de código-fonte quanto a estas métricas.

Emboras as métricas **RFC**, **ACC** e **NOC** não alcançaram os valores mais recomendados pelas configurações "OpenJDK8 Metrics" e "Tomcat Metrics", acredita-se que estas estão ainda em patamares considerado aceitáveis segundo as configurações avaliadas.

Com relação a métrica **LCOM4**, observa-se que tanto para "OpenJDK8 Metrics" quanto para "Tomcat Metrics", houve uma variação no valor percentil que pode indicar que os métodos estão perdendo em coesão, justificando assim a mudança da avaliação de "Bom" para "Regular".

Em contraponto a métrica **LCOM4**, observa-se que a métrica **NOM** melhorou a avaliação, quando considera-se a configuração "OpenJDK8 Metrics", de um intervalo qualitativo "Bom" para "Excelente" podendo assim ser um indício de que as classes estão diminuindo em número de métodos. Em outra avaliação com os resultados obtidos com a métrica **NOM**, observa-se o impacto da avaliação com configurações diferenciadas, onde nos 7 primeiros resultados obtidos com a métrica **NOM** obtém-se "Excelente" para a configuração "OpenJDK8 Metrics" e "Bom" para "Tomcat Metrics". Este fato mostra que a configuração escolhida para a avaliação das métricas de código-fonte pode mudar o parâmetro de medição. Dessa forma, recomenda-se escolher sempre um padrão de medição que seja compatível com as funções e natureza da aplicação avaliada.

Observando os valores percentis obtidos para métrica **CBO**, verifica-se, para ambas as configurações, o intervalo qualitativo "Ruim". Embora o valor percentil alto possa ser um indicativo de que o projeto tem um alto fator de acoplamento entre os objetos, verifica-se que o projeto tem um alto grau de utilização de *frameworks* como Hibernate, VRaptor que é um framework construído sobre o framework Spring. Este fato mostra, portanto, que para a métrica **CBO** é possível considerar o percentil 90 ou avaliar outras configurações que sejam mais adequadas as características do Projeto, pois OpenJDK e Tomcat parecem não ser bom parâmetros de avaliação.

Visando avaliar também a quantidade de consultas OLAP realizadas no ambiente para atender a esse requisito de negócio, foram contabilizadas 2 consultas OLAP de *Drill Down* sobre as dimensões do Projeto, que são as respectivas Figuras 18 e 19.

Com intuito de analisar cada uma das métricas, como se mostra nos gráficos e tabelas do Apêndice B, foram contabilizadas também as operações de análise seletiva: *Slice and Dice* e *Pivoting*. Para cada métrica foram necessários 2 operações *Slice and Dice*; uma para selecionar a configuração e outra para selecionar a métrica desejada e 1

de *Pivoting* para se gerar o gráfico no qual as colunas representam as *releases* do software.

5.2.2 Análise dos Cenários de Limpeza identificados no SICG

Para analisar os cenários de limpeza de código-fonte, foram extraídas as métricas de código-fonte de cada classe e analisadas conforme a Tabela 8. Para cada *release*, foram identificados cenários de limpeza de código-fonte conforme a Figura 20.

Nome do Cenário de Limpeza	Sistema Integrado de Controle e Gestão																		
	OS02	OS04	OS05	OS07	OS08	OS09	OS10	OS11	OS12	OS13	OS14	OS16	OS17	OS19	OS20	OS21	OS22	OS23	OS24
Classe Pouco Coesa	4	9	11	20	26	29	36	47	51	54	55	62	65	66	70	71	73	74	75
Classe com muita Exposição										1	1	1	1	1	1	1	1	2	2
Classe com métodos grandes e/ou muitos condic	1	1		1	3	3	4	4	6	13	15	16	19	22	20	22	22	19	19
Classes com muitos filhos	1	1	1	1	1	1	1	2	2	3	3	5	5	5	5	5	5	5	5
Complexidade Estrutural	15	33	40	64	85	90	108	146	151	158	169	193	201	211	220	226	229	229	230
Interface dos Métodos	6	8	8	8	16	15	24	37	44	48	47	52	56	58	61	64	64	66	66

Figura 20 – Total de Cenários de Limpeza de Código-Fonte identificados por cenário e *Release*

Realizando uma consulta OLAP de *Drill-Up*, obtém-se uma consolidação do número total de cenários de limpeza por cada uma das releases de software analisadas tal como se observa na Figura 21.

Nome do Projeto	OS02	OS04	OS05	OS07	OS08	OS09	OS10	OS11	OS12	OS13	OS14	OS16	OS17	OS19	OS20	OS21	OS22	OS23	OS24
Sistema Integrado de Controle e Gestão	27	52	60	94	131	138	173	236	254	277	290	329	347	363	377	389	394	395	397

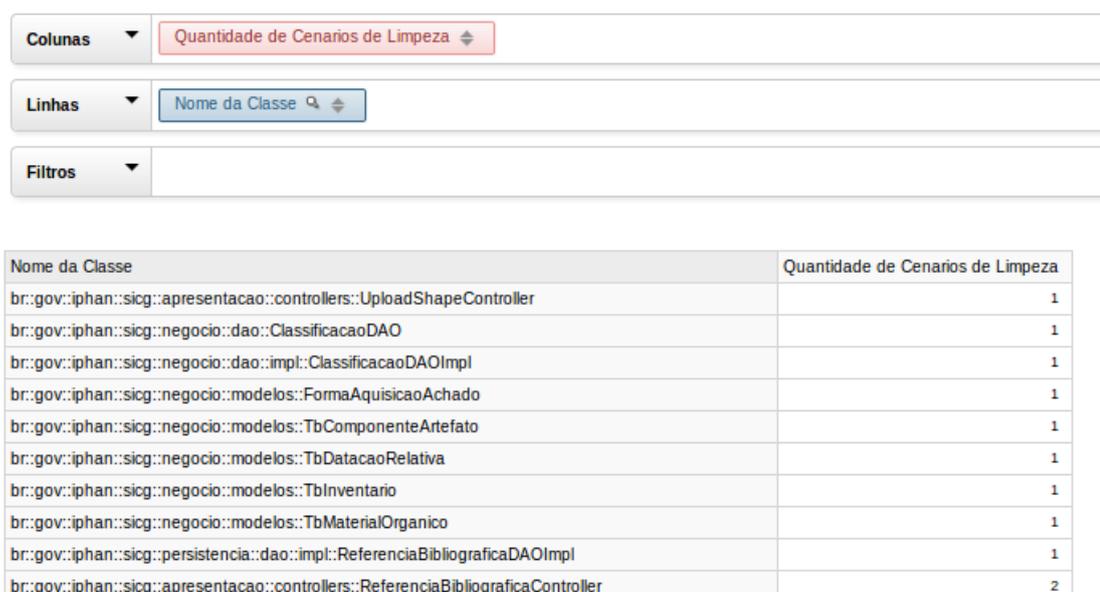
Figura 21 – Total de Cenários de Limpeza de Código-Fonte por Release

Conforme é possível observar nas Figuras 20 e 21, foram detectados mais cenários de limpeza de código-fonte dos tipos **Complexidade Estrutural**, **Classe Pouco Coesa** e **Interface dos Métodos** respectivamente. Os três Cenários de Limpeza com menor número de incidências foram **Classe com Muita Exposição**, **Classe com Muitos Filhos** e **Classe com Métodos Muito Grande e/ou com muitos condicionais**.

No Apêndice C, foram detalhadas, para cada um dos cenários de limpeza de código-fonte, para cada uma das *releases*, as classes que contêm os cenários de limpeza de código-fonte. Para realizar esta consulta em cada um dos cenários, foram realizadas uma consulta OLAP de *Drill-Down* e outra de *Slice and Dice*.

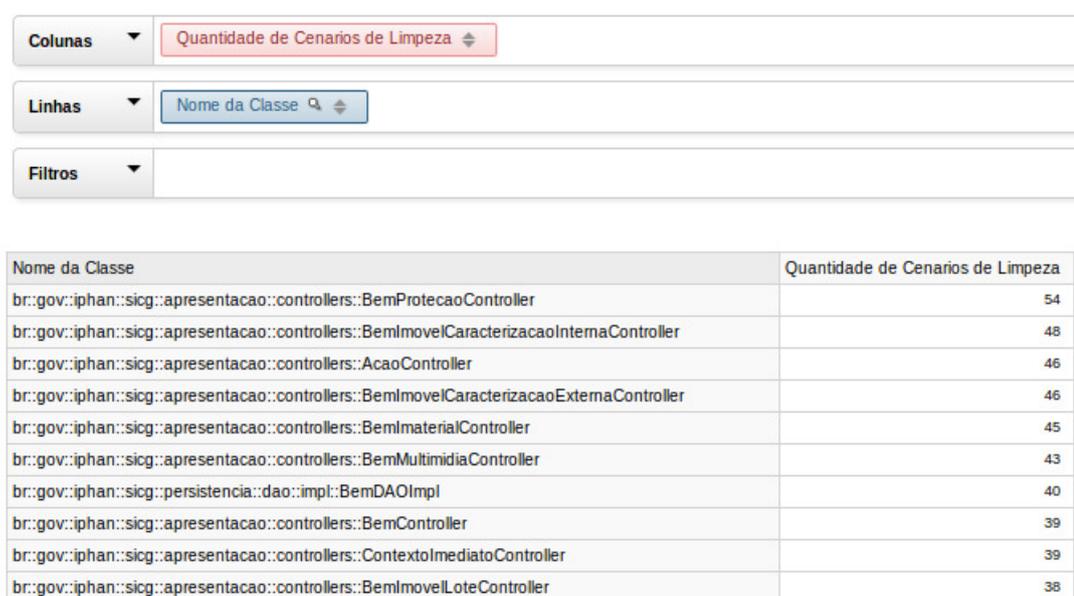
Em uma escala de priorização, de qual cenário de limpeza de código-fonte deve ser tratado primeiro, recomenda-se, para o projeto analisado, tratar o cenário de **Complexidade Estrutural**, pois este sozinho chega a responder entre 55% a 68% da quantidade total de cenários identificados.

Por meio de um *Drill-Down* da consulta OLAP realizada na Figura 20 com mais um *Slice and Dice* foram identificadas as 10 classes que apresentaram a maior quantidade de cenários de limpeza e menor quantidade de cenários de limpeza. Os dados podem ser observados nas Figuras 22 e 23 respectivamente.



Nome da Classe	Quantidade de Cenários de Limpeza
br::gov::iphan::sigg::apresentacao::controllers::UploadShapeController	1
br::gov::iphan::sigg::negocio::dao::ClassificacaoDAO	1
br::gov::iphan::sigg::negocio::dao::impl::ClassificacaoDAOImpl	1
br::gov::iphan::sigg::negocio::modelos::FormaAquisicaoAchado	1
br::gov::iphan::sigg::negocio::modelos::TbComponenteArtefato	1
br::gov::iphan::sigg::negocio::modelos::TbDatacaoRelativa	1
br::gov::iphan::sigg::negocio::modelos::TbInventario	1
br::gov::iphan::sigg::negocio::modelos::TbMaterialOrganico	1
br::gov::iphan::sigg::persistencia::dao::impl::ReferenciaBibliograficaDAOImpl	1
br::gov::iphan::sigg::apresentacao::controllers::ReferenciaBibliograficaController	2

Figura 22 – As 10 classes com menor número de Cénarios de Limpeza



Nome da Classe	Quantidade de Cenários de Limpeza
br::gov::iphan::sigg::apresentacao::controllers::BemProtecaoController	54
br::gov::iphan::sigg::apresentacao::controllers::BemImovelCaracterizacaoInternaController	48
br::gov::iphan::sigg::apresentacao::controllers::AcaoController	46
br::gov::iphan::sigg::apresentacao::controllers::BemImovelCaracterizacaoExternaController	46
br::gov::iphan::sigg::apresentacao::controllers::BemMaterialController	45
br::gov::iphan::sigg::apresentacao::controllers::BemMultimediaController	43
br::gov::iphan::sigg::persistencia::dao::impl::BemDAOImpl	40
br::gov::iphan::sigg::apresentacao::controllers::BemController	39
br::gov::iphan::sigg::apresentacao::controllers::ContextoMediatoController	39
br::gov::iphan::sigg::apresentacao::controllers::BemImovelLoteController	38

Figura 23 – As 10 classes com maior número de Cénarios de Limpeza

5.2.3 Análise da Taxa de Aproveitamento de Oportunidade de Melhoria de Código-Fonte

A partir da identificação dos cenários de limpeza de código-fonte e da contagem do número de classes em uma determinada *release* do software, foi possível calcular a Taxa de Aproveitamento de Oportunidade de Melhoria de Código-Fonte conforme a fórmula 5.1.

$$T_r = \frac{\sum_{i=1}^n Ce_i}{\sum_{i=1}^n Cl_i} \tag{5.1}$$

onde Ce é o total de cenário de limpeza e Cl é a quantidade classes.

Embora se pudesse realizar o cálculo acima, utilizando a consulta que realiza o *Drill-Up* nas dimensões, Figura 21, preferimos consolidar o fato do número de cenários de limpeza de uma determinada release em uma outra tabela fato. Como explicado na Seção 4.3, Kimball e Ross (2002) recomenda-se que medidas de proporção, como taxas, que são fatos não aditivos, devam estar na mesma tabela que o numerador e denominador.

Dessa forma, ao se realizar uma consulta OLAP de *Drill Down* nas dimensões associadas, obteve-se a Taxa de Aproveitamento de Oportunidade de Melhoria de Código-Fonte por cada release do software conforme se mostra nas Figuras 24 e 25 e o crescimento do projeto conforme a Figura 26.

Nome do Projeto	OS	Numero de Classes	Quantidade de Cenarios de Limpeza de Codigo	Taxa de Aproveitamento de Oportunidades de Melhoria de Codigo
Sistema Integrado de Controle e Gestão	OS02	69	27	0,39
	OS04	104	52	0,50
	OS05	125	60	0,48
	OS07	222	94	0,42
	OS08	294	131	0,45
	OS09	300	138	0,46
	OS10	360	173	0,48
	OS11	474	236	0,50
	OS12	532	254	0,48
	OS13	568	277	0,49
	OS14	613	290	0,47
	OS16	730	329	0,45
	OS17	769	347	0,45
	OS19	805	363	0,45
	OS20	861	377	0,44
	OS21	889	389	0,44
	OS22	899	394	0,44
	OS23	908	395	0,44
	OS24	914	397	0,43

Figura 24 – Taxa de Aproveitamento de Oportunidades de Melhoria de Código-Fonte

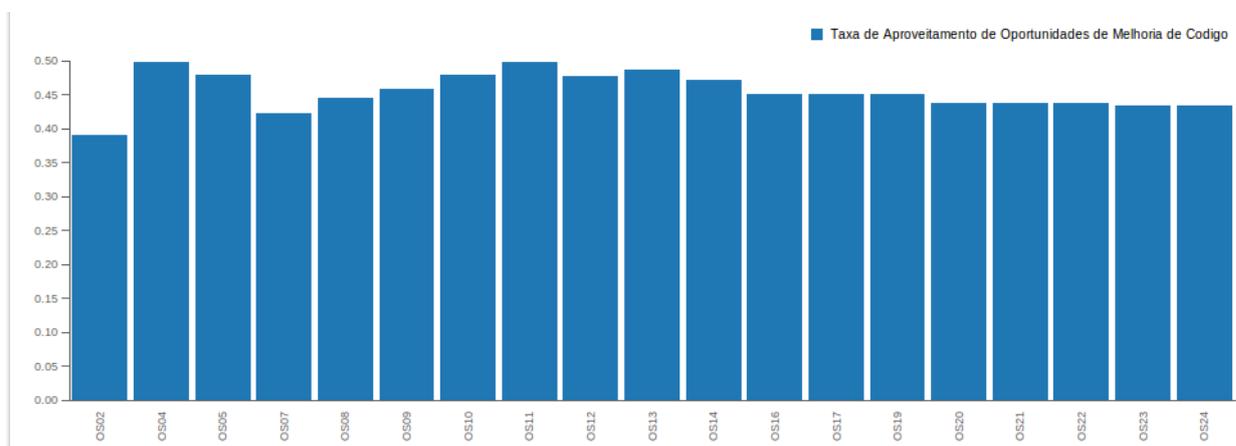


Figura 25 – Gráfico da Taxa de Aproveitamento de Oportunidades de Melhoria de Código-Fonte

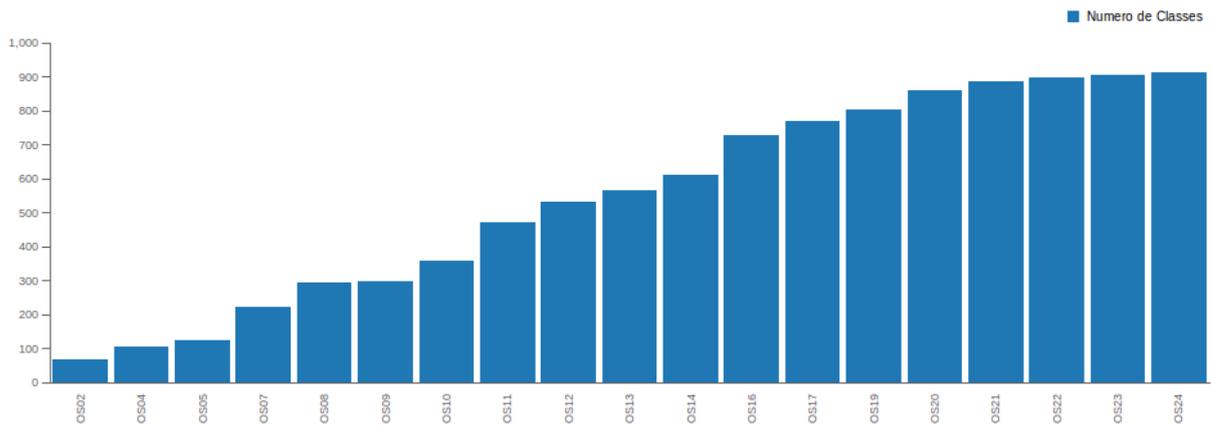


Figura 26 – Crescimento do número de Classes do Projeto SICG

Utilizando as Figuras 24 e 25, observa-se que o projeto cresceu de 67 classes, na primeira release, para 914 classes ao final da última release e o número total de cenários de limpeza de código-fonte identificados cresceu de 27 para 397. Ao se realizar cálculo da Taxa de Aproveitamento de Oportunidade de Melhoria de Código-Fonte, observou-se que há uma tendência de que esta taxa se mantenha entre 0,4 a 0,5. Este fato pode indicar duas hipóteses: a primeira de que o projeto cresceu em uma taxa muito maior que a quantidade de cenários de limpeza, indicando assim uma estabilidade na complexidade do projeto; ou que não foram promovidas atividades de melhoria de código-fonte ao longo das 24 releases.

6 Conclusão

Ao longo deste trabalho de conclusão de curso, foi colocado como objetivo geral a resolução da 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?

Para se responder a esta pesquisa criou-se os objetivos específicos conforme se vê na Tabela 29.

Objetivo Específico
OE1 - Avaliar a Qualidade do Código-Fonte no Projeto
OE2 - Automatizar o processo de medição de métricas de código-fonte.
OE3 - Facilitar a interpretação das métricas de código-fonte.
OE4 - Avaliar indicadores de Código Limpo no Projeto.
OE5 - Avaliar indicadores de aproveitamento de oportunidades de melhoria de Código-Fonte
OE6 - Avaliar quantidade de consultas OLAP sobre as métricas de código-fonte realizadas no ambiente de <i>Data Warehousing</i> em um projeto.

Tabela 29 – Objetivos Específicos do Trabalho

Ao se cumprir a cada um dos objetivos específicos do trabalho, acredita-se cumprir a questão de pesquisa geral, pois os objetivos específicos **OE1**, **OE2**, **OE3** versam sobre a **visibilidade e facilidade de interpretação das métricas de código-fonte** de forma a determinar qual conjunto de métricas pode ser analisado. Uma vez que o conjunto de métricas é escolhido, este pode ter as métricas avaliada segundo configurações de qualidade de métricas de código-fonte, de forma que a coleta e a avaliação ficam automatizados facilitando assim a tomada de decisão de refatoração por parte da equipe de desenvolvimento. Quanto aos objetivos **OE4** e **OE5**, acredita-se apontar mais diretamente para decisão de **refatoração**, pois é possível perceber cenários de limpeza de código-fonte em determinadas classes do código-fonte de forma que a **equipe de desenvolvimento** pode tomar uma decisão de **refatoração do código-fonte**.

O ambiente de *Data Warehousing* permitiu, no estudo de caso da avaliação da qualidade do código-fonte do Sistema Integrado de Conhecimento e Gestão (SICG) do Instituto do Patrimônio Artístico Nacional (IPHAN), avaliar a evolução dos valores percentis do conjunto de métricas de código-fonte neste projeto cumprindo assim o objetivo específico **OE1**.

Comparando-se a análise dos valores percentis obtidos com o ambiente de *data*

warehousing deste trabalho com o relatório de auditoria de código-fonte ¹, obteve-se a Tabela 30.

Características	Este Trabalho (Data Warehousing)	Relatório de Auditoria de Código-Fonte
Ferramenta Utilizada	Analizo	SonarQube
Técnica Estatística Utilizada	Percentil	Média
Métricas Analisadas	LOC, ACC, ACCM, AM-LOC, ANPM, CBO, DIT, LCOM4, NOM, NOC, NPA, RFC	LOC, ACCM
Número de Releases Analisadas	24	1

Tabela 30 – Comparação entre este trabalho e o relatório de auditoria de código-fonte

O relatório de auditoria sobre o código-fonte do SICG foi produzido ao fim do desenvolvimento da 24ª *release* do software. Este utilizou a ferramenta SonarQube, que, como descrito no Capítulo 4, limitou-se a coleta das métricas LOC e ACCM para a linguagem de programação Java. No relatório, chegou-se a conclusão que o projeto não apresentava alto grau de manutenibilidade baseado no valor médio de 1,9 para métrica ACCM e um total de 38.000 linhas de código-fonte.

Ainda sobre a métrica ACCM, ao se comparar com a análise realizada neste trabalho, percebe-se esta métrica apresenta uma medida semelhante entre as ferramentas Analizo e SonarQube. No caso deste, embora este valor não seja retirado do percentil de classes e sim da média, quando avaliado em relação as configurações "Tomcat Metrics" e "OpenJDK Metrics", o valor da aferição, ainda pertence ao intervalo qualitativo "Excelente". Contudo, cabe destacar que apenas uma métrica não é um conjunto representativo de informações no que tange qualidade de código-fonte (MEIRELLES, 2013).

Portanto, acredita-se que ao afirmar que a manutenibilidade do projeto é baixa, baseado apenas na análise de ACCM, incorre-se na possibilidade de emitir um parecer pouco abrangente, o que pode gerar ambiguidades e/ou dúvidas, com relação a qualidade interna do produto. Como mostrado na Seção 5.2.1, a detecção de valores altos de CBO, mesmo quando se considera um alto uso de *frameworks*, precisa ser alvo de uma investigação mais profunda na aplicação, pois os valores altos de CBO podem indicar uma dificuldade futura de manutenibilidade do projeto.

Verificou-se que o ambiente de *Data Warehousing*, atingiu também ao objetivo específico **OE2**, pois foi permitida a automação de todo o processo de medição das métricas

¹ Presente no Repositório deste Trabalho: <<https://github.com/gbrego/TCC>>

de código-fonte, pois foram automatizados desde a coleta, transformação dos dados, avaliação das métricas quanto aos intervalos qualitativos, avaliação dos cenários de limpeza de código-fonte e cálculo da taxa de aproveitamento de oportunidades de melhoria de código-fonte. Na Tabela 31, foram descritos os tempos de execução de cada passo automatizado para a primeira release analisada e a última release analisada.

Tempo	Release 2 (69 classes - 2784 linhas de código)	Release 24 (914 classes - 39.790 linhas de código)
Tempo de Conversão do arquivo CSV resultante do Análise em JSON	6 segundos	10 segundos
Tempo de execução da Extração e Transformação dos Dados na Área de	5 segundos	30 segundos
Tempo execução da avaliação dos Intervalos Percentis das Métricas de Código-Fonte	17 segundos	59 segundos
Tempo execução da avaliação dos Cenários de Limpeza de Código-Fonte	20 segundos	1 minuto e 12 segundos
Tempo de Cálculo da Taxa de Aproveitamento de Oportunidades de Melhoria de Código-Fonte	23 segundos	1 minuto e 29 segundos
Tempo Total	1 minuto e 11 segundos	4 minutos e 20 segundos

Tabela 31 – Tempo de Execução de cada passo automatizado no processo de medição de métricas de código-fonte

O IPHAN não realizou análise das métricas de código-fonte ao longo da execução do contrato. Contudo, entende-se que a utilização do ambiente de *data warehousing* poderia auxiliar a atividade da análise do produto de software recebido. Não é o foco deste trabalho, mas com a utilização de ambiente de *data warehousing* verifica-se a possibilidade de gestores de contrato poderem definir certos níveis de qualidade de código-fonte na execução de contratos de desenvolvimento de software, para contratações de serviços de desenvolvimento de software.

Ao se construir as configurações "OpenJDK Metrics" e "Tomcat Metrics" para os intervalos qualitativos de métricas de código-fonte, percebe-se que foi possível obter diferentes análises sobre as métricas de código-fonte. Logo, é possível inferir a completude do objetivo específico **OE3**, pois foi criado um campo de conhecimento semântico (projeto do *data warehouse*), no qual as métricas podem ser interpretadas sob diversas configurações que podem ser flexibilizadas conforme as necessidades de avaliação e características específicas de cada aplicação avaliada.

Quanto a avaliação de indicadores de código-limpo e acompanhamento da taxa de oportunidades de melhoria de código-fonte, observa-se que se contribuiu com dois indicadores de saúde de bom código-fonte. Isto é, o acompanhamento destes pode permitir à equipe de desenvolvimento ou até a mesmo a gestores de projeto tomarem decisões técnicas mais eficientes no que diz respeito ao código-fonte de um projeto. Estes dois indicadores não só cumprem com objetivos específicos **OE4** e **OE5**, mas contribuem também para se alcançar o **objetivo geral** do trabalho, pois estes são indicadores que podem apoiar uma decisão técnica de refatoração.

Quanto ao objetivo **OE6**, foram contabilizadas, ao longo do trabalho, as consultas OLAP necessárias para responder aos requisitos de negócio para cada um dos processos que foram avaliados conforme a Tabela 32. Cabe salientar que a solução proposta não se limita a quantidade de consultas à apenas a essas analisadas.

Processo a ser Avaliado	Total de Requisitos de Negócio	Consultas OLAP
Avaliação de Valores Percentis de Métricas de Código-Fonte	8 requisitos foram mapeados conforme a Tabela 20.	36 consultas OLAP.
Avaliação de Cenários de Limpeza de Código-Fonte	6 requisitos foram mapeados conforme a Tabela 21.	16 Consultas OLAP.
Acompanhamento de Taxa de Oportunidade de Melhoria de Código-Fonte	1 requisito foi mapeado conforme a Tabela 21.	1 consulta OLAP.

Tabela 32 – Total de Consultas OLAP realizadas

Ao se observar os dados obtidos na Tabela 32 e a própria execução do estudo de caso da Seção 5.2, concluiu-se que o ambiente de *Data Warehousing* conseguiu responder às necessidades estabelecidas para cada um dos requisitos de negócio identificados.

Por meio da utilização do plugin Saiku no ambiente de *Data Warehousing*, é possível que cada uma dessas consultas seja exibida em 8 gráficos diferentes ou em Tabelas. Cada uma das consultas pode ainda conter várias outras consultas OLAP que também responderiam a outras necessidades. Contudo, este trabalho se ateve a contabilizar apenas as que respondiam aos requisitos de negócio levantados.

Por fim, ao se analisar o problema de associação entre as medidas numéricas e interpretações e o problema da ausência de mecanismos de separação, agregação e formas de visualização de dados que permitam a fácil identificação de um ponto de melhoria no código-fonte, verifica-se que estes foram sanados com a utilização do ambiente de *Data*

Warehousing. Neste ambiente, foi possível separar cada uma das métricas, agregá-las por releases, classes e ainda foi possível visualizar pontos de melhorias de melhorias de código-fonte em cada uma classes do projeto em uma determinada *release*.

Portanto, o ambiente de *Data Warehousing* para métricas de código-fonte logrou êxito em aumentar a visibilidade das métricas de código-fonte, facilitando a interpretação destas de modo a apoiar a decisão de refatoração em um projeto. Desta forma, concluiu-se que a questão de pesquisa deste trabalho foi devidamente respondida, assim como os objetivos geral e específicos alcançados.

6.1 Limitações

Como uma limitação do trabalho, cabe destacar que o ambiente de *Data Warehousing*, para analisar métricas de código-fonte, neste trabalho, é dependente da ferramenta Analizo, ferramenta de análise estática de código-fonte escolhida. Esta se limita as linguagens de programação C, C++ e Java, até o momento.

Como apresentado anteriormente, na Seção 5.1.6, a utilização de um estudo de caso por si só não é suficiente para generalizar os resultados dele obtidos (YIN, 2011). Por esse motivo, a utilização do estudo de caso é uma limitação para a generalização do trabalho. Uma outra limitação quanto ao estudo de caso é a falta de utilização do teste de hipótese como forma de validação estatística da hipótese H0.

Quanto aos metadados do Projeto de *Data Warehouse*, apresentados 4.3, verifica-se que para mapear corretamente um determinado cenário de código-fonte com um intervalo de uma métrica, escolheu-se como uma decisão de projeto colocar duas chaves estrangeiras da Tabela "Meta_Metric_Ranges" na Tabela "Meta_Metric_Ranges_Meta_Scenario". Este fato implica uma limitação do modelo em que não é possível associar mais de dois intervalos de duas métricas de código-fonte a um cenário de limpeza de código-fonte.

Quanto a eficácia do modelo do ponto de vista de usuário, este trabalhou não verificou a utilização do ambiente de *data warehousing* por parte do IPHAN. Sugere-se um outro trabalho que contenha um estudo de caso para se medir a eficiência e a eficácia do projeto de *data warehouse* no ambiente do IPHAN.

6.2 Trabalhos Futuros

Quanto aos trabalhos futuros, foi possível identificar duas possibilidades imediatas. A primeira possibilidade é uma melhor avaliação de como os padrões de projeto, propostos inicialmente na Tabela 8, se correlacionam com as avaliações das métricas de código-fonte e a detecção dos cenários de limpeza de código-fonte. A segunda possibilidade é a investigação, com mecanismos estatísticos, com apoio de estudo de múltiplos casos, de

como que a Taxa de Aproveitamento de Oportunidades de Melhoria de código-fonte pode se comportar a depender de como se utiliza as boas práticas de código-fonte citadas na Tabela 7.

Referências

- 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 18 e 21.
- 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 25.
- BECK, K. *Smalltalk Best Practice Patterns. Volume 1: Coding*. [S.l.]: Prentice Hall, Englewood Cliffs, NJ, 1997. Citado na página 26.
- BECK, K. Embracing change with extreme programming. *Computer*, v. 32, n. 10, p. 70–77, 1999. ISSN 0018-9162. Citado na página 50.
- BECK, K. *Extreme Programming Explained*. [S.l.]: Addison Wesley, 1999. Citado na página 17.
- BECK, K. *Test-driven development: by example*. [S.l.]: Addison-Wesley Professional, 2003. Citado na página 17.
- BECK, K. *Implementation patterns*. [S.l.]: Pearson Education, 2007. Citado 3 vezes nas páginas 17, 30 e 32.
- BECKER, K. et al. Spdw: A software development process performance data warehousing environment. In: *Proceedings of the 30th Annual IEEE/NASA Software Engineering Workshop*. Washington, DC, USA: IEEE Computer Society, 2006. (SEW '06), p. 107–118. ISBN 0-7695-2624-1. Disponível em: <<http://dx.doi.org/10.1109/SEW.2006.31>>. Citado 2 vezes nas páginas 19 e 60.
- BRERETON, P. et al. Using a protocol template for case study planning. In: *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering. University of Bari, Italy*. [S.l.: s.n.], 2008. Citado na página 60.
- CASTELLANOS, M. et al. ibom: A platform for intelligent business operation management. In: *Proceedings of the 21st International Conference on Data Engineering*. Washington, DC, USA: IEEE Computer Society, 2005. (ICDE '05), p. 1084–1095. ISBN 0-7695-2285-8. Disponível em: <<http://dx.doi.org/10.1109/ICDE.2005.73>>. Citado 2 vezes nas páginas 19 e 60.
- 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 34 e 40.
- CHIDAMBER, S. R.; KEMERER, C. F. A Metrics Suite for Object-Oriented Design. *IEEE Transactions on Software Engineering*, v. 20, n. 6, p. 476–493, 1994. Citado na página 25.
- CHULANI, S. et al. Metrics for managing customer view of software quality. In: *Proceedings of the 9th International Symposium on Software Metrics*. Washington, DC,

USA: IEEE Computer Society, 2003. (METRICS '03), p. 189–. ISBN 0-7695-1987-3. Disponível em: <<http://dl.acm.org/citation.cfm?id=942804.943748>>. Citado na página 18.

CMMI. *CMMI® for Development, Version 1.3*. [S.l.], 2010. Disponível em: <<http://www.sei.cmu.edu/library/abstracts/reports/10tr033.cfm>>. Citado na página 21.

CODD, E. F.; CODD, S. B.; SALLEY, C. T. *Providing OLAP (On-Line Analytical Processing) to User-Analysis: An IT Mandate*. [S.l.]: E. F. Codd & Associates, 1993. Citado na página 39.

EMANUELSSON, P.; NILSSON, U. A comparative study of industrial static analysis tools. *Electron. Notes Theor. Comput. Sci.*, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 217, p. 5–21, jul. 2008. ISSN 1571-0661. Citado na página 17.

FENTON, N. E.; PFLEEGER, S. L. *Software Metrics: A Rigorous and Practical Approach*. 2 edition. ed. [S.l.]: Course Technology, 1998. 656 p. Citado 3 vezes nas páginas 21, 22 e 23.

FOLLECO, A. et al. Learning from software quality data with class imbalance and noise. In: *Proceedings of the Nineteenth International Conference on Software Engineering & Knowledge Engineering (SEKE 2007), Boston, Massachusetts, USA, July 9-11, 2007*. [S.l.]: Knowledge Systems Institute Graduate School, 2007. p. 487. ISBN 1-891706-20-9. Citado 2 vezes nas páginas 19 e 60.

FONSECA, R.; SIMOES, A. Alternativas ao xml: Yaml e json. 2007. Citado na página 86.

FOWLER, M. *Refactoring: improving the design of existing code*. [S.l.]: Addison-Wesley Professional, 1999. Citado na página 17.

GAMMA, E. et al. *Design patterns: elements of reusable object-oriented software*. [S.l.]: Pearson Education, 1994. Citado na página 32.

GARDNER, S. R. Building the. *Communications of the ACM*, v. 41, n. 9, p. 53, 1998. Citado na página 34.

GOLFARELLI, M.; MAIO, D.; RIZZI, S. Conceptual Design of Data Warehouses from E/R Schemes. 1998. Citado 4 vezes nas páginas 9, 38, 39 e 51.

GOPAL, A.; MUKHOPADHYAY, T.; KRISHNAN, M. S. The impact of institutional forces on software metrics programs. *IEEE Trans. Softw. Eng.*, IEEE Press, Piscataway, NJ, USA, v. 31, n. 8, p. 679–694, ago. 2005. ISSN 0098-5589. Disponível em: <<http://dx.doi.org/10.1109/TSE.2005.95>>. Citado na página 34.

HARMAN, M. Why source code analysis and manipulation will always be important. In: *IEEE. Source Code Analysis and Manipulation (SCAM), 2010 10th IEEE Working Conference on*. [S.l.], 2010. p. 7–19. Citado na página 24.

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 25.

- INMON, W. H. *Building the Data Warehouse*. New York, NY, USA: John Wiley & Sons, Inc., 1992. ISBN 0471569607. Citado 3 vezes nas páginas 36, 38 e 40.
- ISO/IEC 15939. *ISO/IEC 15939: Software Engineering - Software Measurement Process*. [S.l.], 2002. Citado 4 vezes nas páginas 11, 21, 22 e 23.
- ISO/IEC 25023. *ISO/IEC 25023: Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQaRE) - Measurement of system and software product quality*. [S.l.], 2011. Citado 3 vezes nas páginas 9, 17 e 23.
- JONES, T. C. *Applied Software Measurement: Assuring Productivity and Quality*. New York: McGraw-Hill, 1991. Citado na página 24.
- 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 11 vezes nas páginas 9, 35, 36, 37, 38, 39, 40, 47, 50, 54 e 73.
- LEHMAN, M. M. Programs, life cycles, and laws of software evolution. *Proc. IEEE*, v. 68, n. 9, p. 1060–1076, September 1980. Citado na página 24.
- LI, W.; HENRY, S. Object-oriented metrics that predict maintainability. *Journal of Systems and Software*, v. 23, n. 2, p. 111–122, nov. 1993. ISSN 01641212. Disponível em: <<http://www.sciencedirect.com/science/article/pii/016412129390077B>>. Citado na página 25.
- LORENZ, M.; KIDD, J. *Object-Oriented Software Metrics*. [S.l.]: Prentice Hall, 1994. Citado na página 26.
- MACHINI, J. a. et al. *Código Limpo e seu Mapeamento para Métricas de Código-Fonte*. [S.l.]: Universidade de São Paulo, 2010. Citado 4 vezes nas páginas 11, 30, 31 e 32.
- 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 17 e 32.
- MARINESCU, R.; RATIU, D. Quantifying the quality of object-oriented design: The factor-strategy model. In: IEEE. *Reverse Engineering, 2004. Proceedings. 11th Working Conference on*. [S.l.], 2004. p. 192–201. Citado 2 vezes nas páginas 17 e 32.
- 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 2 vezes nas páginas 30 e 32.
- MCCABE, T. J. A Complexity Measure. *IEEE Transactions Software Engineering*, v. 2, n. 4, p. 308–320, December 1976. Citado na página 24.
- MCCABE, T. J.; DREYER, L. A.; WATSON, A. H. Testing An Object-Oriented Application. *Journal of the Quality Assurance Institute*, v. 8, n. 4, p. 21–27, October 1994. Citado na página 25.
- MEIRELLES, P. R. M. *Monitoramento de métricas de código-fonte em projetos de software livre*. Tese (Doutorado) — Instituto de Matemática e Estatística – Universidade de São Paulo (IME/USP), 2013. Citado 10 vezes nas páginas 11, 17, 22, 23, 24, 26, 27, 28, 46 e 76.

- MILLS, E. E. Metrics in the software engineering curriculum. *Ann. Softw. Eng.*, J. C. Baltzer AG, Science Publishers, Red Bank, NJ, USA, v. 6, n. 1-4, p. 181–200, abr. 1999. ISSN 1022-7091. Citado 2 vezes nas páginas 23 e 24.
- MINARDI, R. C. de M. *Visualização de Dados*. 2013. Universidade Federal de Minas Gerais - UFMG. Disponível em: <<http://homepages.dcc.ufmg.br/~raquelcm/material/visualizacao/aulas/>>. Citado na página 42.
- MOHA, N. et al. Decor: A method for the specification and detection of code and design smells. *Software Engineering, IEEE Transactions on*, IEEE, v. 36, n. 1, p. 20–36, 2010. Citado 2 vezes nas páginas 17 e 32.
- MOHA, N.; GUÉHÉNEUC, Y.-G.; LEDUC, P. Automatic generation of detection algorithms for design defects. In: IEEE. *Automated Software Engineering, 2006. ASE'06. 21st IEEE/ACM International Conference on*. [S.l.], 2006. p. 297–300. Citado 2 vezes nas páginas 17 e 32.
- MOHA, N. et al. A domain analysis to specify design defects and generate detection algorithms. In: *Fundamental Approaches to Software Engineering*. [S.l.]: Springer, 2008. p. 276–291. Citado 2 vezes nas páginas 17 e 32.
- 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 2 vezes nas páginas 11 e 39.
- NIELSON, F.; NIELSON, H. R.; HANKIN, C. *Principles of Program Analysis*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1999. ISBN 3540654100. Citado na página 17.
- PALZA, E.; FUHRMAN, C.; ABRAN, A. Establishing a generic and multidimensional measurement repository in cmmi context. In: IEEE. *Software Engineering Workshop, 2003. Proceedings. 28th Annual NASA Goddard*. [S.l.], 2003. p. 12–20. Citado 2 vezes nas páginas 19 e 60.
- RAO, A. A.; REDDY, K. N. Detecting bad smells in object oriented design using design change propagation probability matrix 1. Citeseer, 2007. Citado 2 vezes nas páginas 17 e 32.
- ROCHA, A. B. *Guardando Históricos de Dimensões em Data Warehouses*. Universidade Federal da Paraíba - Centro de Ciências e Tecnologia: [s.n.], 2000. Citado 5 vezes nas páginas 11, 34, 39, 40 e 41.
- ROSENBERG, L. H.; HYATT, L. E. Software Quality Metrics for Object-Oriented Environments. *Crosstalk - the Journal of Defense Software Engineering*, v. 10, 1997. Citado na página 26.
- 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 2 vezes nas páginas 19 e 60.

- SAMPAIO, M. C. *Projeto Conceitual de Data Warehouse*. 2007. Universidade Federal de Campina Grande - UFCG. Disponível em: <<http://dsc.ufcg.edu.br/~sampaio/cursos/2007.1/PosGraduacao/BDMultidimensional/II-ProjetoConceitual/>>. Citado 2 vezes nas páginas 9 e 39.
- SANTOS, H. R. N. Aline Gonçalves dos. *Adoção de Métodos Ágeis e do Pensamento Lean na Gestão de Contratos de Fornecedores de Desenvolvimento de Software em Organizações Públicas Brasileiras: um Estudo de Caso*. [S.l.]: Universidade de Brasília, 2014. Citado na página 62.
- SHARBLE, R.; COHEN, S. The Object-Oriented Brewery: A Comparison of Two Object-Oriented Development Methods. *Software Engineering Notes*, v. 18, n. 2, p. 60–73, 1993. Citado na página 26.
- SHIH, T. et al. Decomposition of Inheritance Hierarchy DAGs for Object-Oriented Software Metrics. In: *Workshop on Engineering of Computer-Based Systems (ECBS 97)*. [S.l.: s.n.], 1997. p. 238. Citado na página 25.
- SILVEIRA, P. S.; BECKER, K.; RUIZ, D. D. Spdw+: a seamless approach for capturing quality metrics in software development environments. *Software Quality Control*, Kluwer Academic Publishers, Hingham, MA, USA, v. 18, n. 2, p. 227–268, jun. 2010. ISSN 0963-9314. Disponível em: <<http://dx.doi.org/10.1007/s11219-009-9092-9>>. Citado 3 vezes nas páginas 19, 34 e 60.
- SOMMERVILLE, I. *Software Engineering*. 9. ed. Harlow, England: Addison-Wesley, 2010. ISBN 978-0-13-703515-1. Citado na página 17.
- SPOFFORD, G. et al. *MDX Solutions with Microsoft SQL Server Analysis Services 2005 and Hyperion Essbase*. [S.l.]: Wiley Pub., 2006. Citado na página 58.
- TIMES, V. C. *Sistemas de DW*. 2012. Universidade Federal de Pernambuco - UFPE. Disponível em: <www.cin.ufpe.br/~if695/bda_dw.pdf>. Citado 6 vezes nas páginas 9, 11, 36, 37, 39 e 41.
- WICHMANN, B. et al. Industrial perspective on static analysis. *Software Engineering Journal*, 1995. Citado na página 17.
- WOHLIN, C. et al. *Experimentation in software engineering*. [S.l.]: Springer, 2012. Citado 2 vezes nas páginas 9 e 60.
- YAMASHITA, A. Assessing the capability of code smells to explain maintenance problems: an empirical study combining quantitative and qualitative data. *Empirical Software Engineering*, Springer, p. 1–33, 2013. Citado na página 17.
- YIN, R. K. *Applications of case study research*. [S.l.]: Sage, 2011. Citado 3 vezes nas páginas 65, 66 e 79.

APÊNDICE A – Descrição do Processo de ETL no Kettle

Neste apêndice, será apresentado a implementação do ETL no Kettle, onde se utilizou dos arquivos CSV resultantes da análise de métricas de código-fonte do Analizo. Embora, o Kettle tivesse componentes de interpretação dos elementos de CSV, no presente trabalho, decidiu-se por converter CSV obtido do Analizo em arquivos JSON, visto que componente de CSV do Kettle, converte-o para XML, sendo que este é mais lento e menos versátil quando comparado ao JSON, tal como se mostra no trabalho de [Fonseca e Simoes \(2007\)](#). Visando realizar a conversão, foi escrito um pequeno *parser* na linguagem *Ruby*, tal como se vê no Código-Fonte 2.

```
1 #!/usr/bin/env ruby
2 require 'csv'
3 require 'json'
4
5 if ARGV.size != 2
6   puts 'Forma de Utilizar:
7         parserCSVJSON input_file.csv
8         output_file.json'
9   exit(1)
10 end
11
12 lines = CSV.open(ARGV[0]).readlines
13 keys = lines.delete lines.first
14
15 File.open(ARGV[1], 'w') do |f|
16   data = lines.map do |values|
17     Hash[keys.zip(values)]
18   end
19   f.puts JSON.pretty_generate(data)
20 end
```

Código-Fonte 2 – *Parser* de CSV para JSON

A.1 Implementações das *Transformations*

Como explicado anteriormente, o Kettle utiliza o componente de *Transformation* para realizar cálculos, consultas em tabelas, inserções em tabelas, leitura de dados e entre outros. Alguns desses componentes são mostrados na Figura 27.



Figura 27 – Componentes do Kettle que foram utilizadas nas Transformações

O componente *JSON Input* serve para ler os dados provenientes de um arquivo JSON, em que o conteúdo de cada variável pode ser lido como: `$.@nome_da_variavel`. O componente *Univariate Statistics* é utilizado para se calcular estatísticas como média, mediana, número de amostras e percentis, que foram utilizados no cálculo dos intervalos percentis das métricas de código-fonte; O componente *Execute SQL Script* foi utilizado para recuperação de dados no Metadados e Dimensões e também para inserção de nas Tabelas Fatos; O componente *Combination Lookup* foi utilizado para se verificar se um determinado já existia em uma Dimensão, caso existisse, apenas se retornava o id da tábua, se não o inseria na Dimensão; Por fim o componente *Select Values* foi utilizado para filtrar os dados proveniente de outros componentes em uma *Transformation*.

Na primeira transformação, como se mostra na Figura 28, obtém-se os dados provenientes do arquivo JSON com componente *JSON Input*. Em uma primeira etapa, coletava-se sobre dados: nome do projeto, release do arquivo, data de lançamento da release. Após a coleta dos dados do arquivo JSON, se inseria, conforme as verificações do componente *Combination Lookup*, nas dimensões correspondentes.

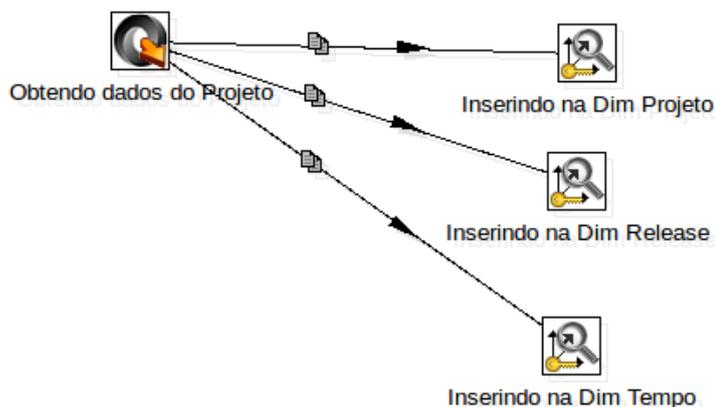


Figura 28 – Primeira Transformação realizada no Kettle

Na segunda transformação, como se mostra na Figura 29, cobre-se o processo de negócio de avaliação dos valores percentis das métricas de código-fonte do projeto em uma determinada *release* do software. Para tal, foi coletado os valores das métricas de código-fonte de cada classe com o componente *JSON Input*. Após a coleta dos valores das

métricas, esses eram direcionados ao componente *Univariate Statistics* que realiza cálculos estatísticos. Após a realização dos cálculos, foi realizado um filtro com *Select Values* a fim de se obter apenas os percentis obtidos para cada uma das métricas.

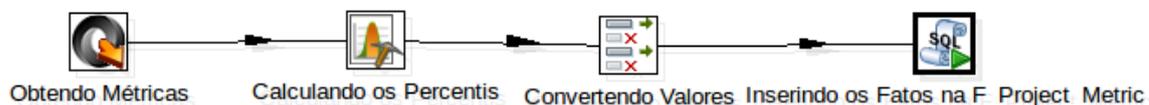


Figura 29 – Segunda Transformação realizada no Kettle

Por fim, foi realizado a avaliação dos valores percentis, em intervalos qualitativos nos metadados com o componente *Execute SQL Script*. Neste componente, recebeu-se o código-fonte descrito no Código-Fonte 3, onde cada foi substituído por uma variável dentro da *Transformation*.

```

1 USE source_info;
2
3 SET @idProject = (SELECT max(idProject) from D_Project);
4
5 SET @idTime = (SELECT max(idTime) from D_Time);
6
7 SET @idRelease = (SELECT max(idRelease) from D_Release);
8
9 SET @Project_Language = (SELECT project_language from D_Project);
10
11 SET @Best_Configuration = (SELECT idConfiguration FROM D_Configuration
12     where configuration_name like '%Open JDK8 Metrics%' and
13     language_name LIKE CONCAT('%', @Project_Language, '%'));
14 SET @Worst_Configuration = (SELECT idConfiguration FROM D_Configuration
15     where configuration_name like '%Tomcat Metrics%' and
16     language_name LIKE CONCAT('%', @Project_Language, '%'));
17 # Consulta da Metrica LOC
18 SET @idLOC = (SELECT idMetric FROM D_Metric where metric_abbreviation='
19     LOC');
20 # Consulta de Indicador de Qualidade de LOC na Configuracao Open JDK8
21     Metrics
22 SET @qualityBestLOC = (SELECT idQuality FROM Meta_Metric_Ranges
23     INNER JOIN D_Quality
24     ON Meta_Metric_Ranges.quality_index=D_Quality.quality_index where
25     language_name = @Project_Language

```

```
24 and metric_name='LOC' and ? <= max AND ? >= min and configuration_name
    like '%Open JDK8 Metrics%');
25
26 # Consulta de Indicador de Qualidade de LOC na Configuracao Tomcat
    Metrics
27 SET @qualityWorstLOC = (SELECT idQuality FROM Meta_Metric_Ranges
28 INNER JOIN D_Quality
29 ON Meta_Metric_Ranges.quality_index=D_Quality.quality_index where
    language_name = @Project_Language
30 and metric_name='LOC' and ? <= max AND ? >= min and configuration_name
    like '%Tomcat Metrics%');
31
32 # Inserindo o Fato do LOC na Configuracao Open JDK8 Metrics ;
33 INSERT INTO 'F_Project_Metric'
34 ('percentil_value', 'D_Project_idProject', 'D_Metric_idMetric', '
    D_Quality_idQuality',
35 'D_Configuration_idConfiguration', 'D_Release_idRelease', 'D_Time_idTime
    ')
36 VALUES
37 (?, @idProject, @idLOC, @qualityBestLOC, @Best_Configuration, @idRelease
    , @idTime);
38
39 # Inserindo o Fato do LOC na Configuracao Tomcat Metrics ;
40 INSERT INTO 'F_Project_Metric'
41 ('percentil_value', 'D_Project_idProject', 'D_Metric_idMetric', '
    D_Quality_idQuality',
42 'D_Configuration_idConfiguration', 'D_Release_idRelease', 'D_Time_idTime
    ')
43 VALUES
44 (?, @idProject, @idLOC, @qualityWorstLOC, @Worst_Configuration,
    @idRelease, @idTime);
45
46 # Consulta da Metrica ACCM
47 SET @idACCM = (SELECT idMetric FROM D_Metric where metric_abbreviation='
    ACCM');
48
49 # Consulta de Indicador de Qualidade de ACCM na Configuracao Open JDK8
    Metrics
50 SET @qualityBestACCM = (SELECT idQuality FROM Meta_Metric_Ranges
51 INNER JOIN D_Quality
52 ON Meta_Metric_Ranges.quality_index=D_Quality.quality_index where
    language_name = @Project_Language
53 and metric_name='ACCM' and ? <= max AND ? >= min and configuration_name
    like '%Open JDK8 Metrics%');
54
55 # Consulta de Indicador de Qualidade de ACCM na Configuracao Tomcat
    Metrics
```

```
56 SET @qualityWorstACCM = (SELECT idQuality FROM Meta_Metric_Ranges
57 INNER JOIN D_Quality
58 ON Meta_Metric_Ranges.quality_index=D_Quality.quality_index where
    language_name = @Project_Language
59 and metric_name='ACCM' and ? <= max AND ? >= min and configuration_name
    like '%Tomcat Metrics%');
60
61 # Inserindo o Fato do ACCM na Configuracao Open JDK8 Metrics ;
62 INSERT INTO 'F_Project_Metric '
63 ('percentil_value', 'D_Project_idProject', 'D_Metric_idMetric', '
    D_Quality_idQuality',
64 'D_Configuration_idConfiguration', 'D_Release_idRelease','D_Time_idTime
    ')
65 VALUES
66 (?, @idProject, @idACCM, @qualityBestACCM, @Best_Configuration,
    @idRelease, @idTime);
67
68 # Inserindo o Fato do ACCM na Configuracao Tomcat Metrics ;
69 INSERT INTO 'F_Project_Metric '
70 ('percentil_value', 'D_Project_idProject', 'D_Metric_idMetric', '
    D_Quality_idQuality',
71 'D_Configuration_idConfiguration', 'D_Release_idRelease','D_Time_idTime
    ')
72 VALUES
73 (?, @idProject, @idACCM, @qualityWorstACCM, @Worst_Configuration,
    @idRelease, @idTime);
74
75 # Consulta da Metrica AMLOC
76 SET @idAMLOC = (SELECT idMetric FROM D_Metric where metric_abbreviation=
    'AMLOC');
77
78 # Consulta de Indicador de Qualidade de AMLOC na Configuracao Open JDK8
    Metrics
79 SET @qualityBestAMLOC = (SELECT idQuality FROM Meta_Metric_Ranges
80 INNER JOIN D_Quality
81 ON Meta_Metric_Ranges.quality_index=D_Quality.quality_index where
    language_name = @Project_Language
82 and metric_name='AMLOC' and ? <= max AND ? >= min and configuration_name
    like '%Open JDK8 Metrics%');
83
84 # Consulta de Indicador de Qualidade de AMLOC na Configuracao Tomcat
    Metrics
85 SET @qualityWorstAMLOC = (SELECT idQuality FROM Meta_Metric_Ranges
86 INNER JOIN D_Quality
87 ON Meta_Metric_Ranges.quality_index=D_Quality.quality_index where
    language_name = @Project_Language
88 and metric_name='AMLOC' and ? <= max AND ? >= min and configuration_name
```

```
        like '%Tomcat Metrics%');
89
90 # Inserindo o Fato do AMLOC na Configuracao Open JDK8 Metrics ;
91 INSERT INTO 'F_Project_Metric'
92 ('percentil_value', 'D_Project_idProject', 'D_Metric_idMetric', '
    D_Quality_idQuality',
93 'D_Configuration_idConfiguration', 'D_Release_idRelease', 'D_Time_idTime
    ')
94 VALUES
95 (?, @idProject, @idAMLOC, @qualityBestAMLOC, @Best_Configuration,
    @idRelease, @idTime);
96
97 # Inserindo o Fato do AMLOC na Configuracao Tomcat Metrics ;
98 INSERT INTO 'F_Project_Metric'
99 ('percentil_value', 'D_Project_idProject', 'D_Metric_idMetric', '
    D_Quality_idQuality',
100 'D_Configuration_idConfiguration', 'D_Release_idRelease', 'D_Time_idTime
    ')
101 VALUES
102 (?, @idProject, @idAMLOC, @qualityWorstAMLOC, @Worst_Configuration,
    @idRelease, @idTime);
103
104 # Consulta da Metrica ACC
105 SET @idACC = (SELECT idMetric FROM D_Metric where metric_abbreviation='
    ACC');
106
107 # Consulta de Indicador de Qualidade de ACC na Configuracao Open JDK8
    Metrics
108 SET @qualityBestACC = (SELECT idQuality FROM Meta_Metric_Ranges
109 INNER JOIN D_Quality
110 ON Meta_Metric_Ranges.quality_index=D_Quality.quality_index where
    language_name = @Project_Language
111 and metric_name='ACC' and ? <= max AND ? >= min and configuration_name
    like '%Open JDK8 Metrics%');
112
113 # Consulta de Indicador de Qualidade de ACC na Configuracao Tomcat
    Metrics
114 SET @qualityWorstACC = (SELECT idQuality FROM Meta_Metric_Ranges
115 INNER JOIN D_Quality
116 ON Meta_Metric_Ranges.quality_index=D_Quality.quality_index where
    language_name = @Project_Language
117 and metric_name='ACC' and ? <= max AND ? >= min and configuration_name
    like '%Tomcat Metrics%');
118
119 # Inserindo o Fato do ACC na Configuracao Open JDK8 Metrics ;
120 INSERT INTO 'F_Project_Metric'
121 ('percentil_value', 'D_Project_idProject', 'D_Metric_idMetric', '
    ')
```

```
        D_Quality_idQuality',
122 'D_Configuration_idConfiguration', 'D_Release_idRelease', 'D_Time_idTime
    ')
123 VALUES
124 (?, @idProject, @idACC, @qualityBestACC, @Best_Configuration, @idRelease
    , @idTime);
125
126 # Inserindo o Fato do ACC na Configuracao Tomcat Metrics ;
127 INSERT INTO 'F_Project_Metric'
128 ('percentil_value', 'D_Project_idProject', 'D_Metric_idMetric', '
    D_Quality_idQuality',
129 'D_Configuration_idConfiguration', 'D_Release_idRelease', 'D_Time_idTime
    ')
130 VALUES
131 (?, @idProject, @idACC, @qualityWorstACC, @Worst_Configuration,
    @idRelease, @idTime);
132
133 # Consulta da Metrica ANPM
134 SET @idANPM = (SELECT idMetric FROM D_Metric where metric_abbreviation='
    ANPM');
135
136 # Consulta de Indicador de Qualidade de ANPM na Configuracao Open JDK8
    Metrics
137 SET @qualityBestANPM = (SELECT idQuality FROM Meta_Metric_Ranges
138 INNER JOIN D_Quality
139 ON Meta_Metric_Ranges.quality_index=D_Quality.quality_index where
    language_name = @Project_Language
140 and metric_name='ANPM' and ? <= max AND ? >= min and configuration_name
    like '%Open JDK8 Metrics%');
141
142 # Consulta de Indicador de Qualidade de ANPM na Configuracao Tomcat
    Metrics
143 SET @qualityWorstANPM = (SELECT idQuality FROM Meta_Metric_Ranges
144 INNER JOIN D_Quality
145 ON Meta_Metric_Ranges.quality_index=D_Quality.quality_index where
    language_name = @Project_Language
146 and metric_name='ANPM' and ? <= max AND ? >= min and configuration_name
    like '%Tomcat Metrics%');
147
148 # Inserindo o Fato do ANPM na Configuracao Open JDK8 Metrics ;
149 INSERT INTO 'F_Project_Metric'
150 ('percentil_value', 'D_Project_idProject', 'D_Metric_idMetric', '
    D_Quality_idQuality',
151 'D_Configuration_idConfiguration', 'D_Release_idRelease', 'D_Time_idTime
    ')
152 VALUES
153 (?, @idProject, @idANPM, @qualityBestANPM, @Best_Configuration,
```

```
        @idRelease, @idTime);
154
155 # Inserindo o Fato do ANPM na Configuracao Tomcat Metrics ;
156 INSERT INTO 'F_Project_Metric'
157 ('percentil_value', 'D_Project_idProject', 'D_Metric_idMetric', '
    D_Quality_idQuality',
158 'D_Configuration_idConfiguration', 'D_Release_idRelease', 'D_Time_idTime
    ')
159 VALUES
160 (?, @idProject, @idANPM, @qualityWorstANPM, @Worst_Configuration,
    @idRelease, @idTime);
161
162 # Consulta da Metrica CBO
163 SET @idCBO = (SELECT idMetric FROM D_Metric where metric_abbreviation='
    CBO');
164
165 # Consulta de Indicador de Qualidade de CBO na Configuracao Open JDK8
    Metrics
166 SET @qualityBestCBO = (SELECT idQuality FROM Meta_Metric_Ranges
167 INNER JOIN D_Quality
168 ON Meta_Metric_Ranges.quality_index=D_Quality.quality_index where
    language_name = @Project_Language
169 and metric_name='CBO' and ? <= max AND ? >= min and configuration_name
    like '%Open JDK8 Metrics%');
170
171 # Consulta de Indicador de Qualidade de CBO na Configuracao Tomcat
    Metrics
172 SET @qualityWorstCBO = (SELECT idQuality FROM Meta_Metric_Ranges
173 INNER JOIN D_Quality
174 ON Meta_Metric_Ranges.quality_index=D_Quality.quality_index where
    language_name = @Project_Language
175 and metric_name='CBO' and ? <= max AND ? >= min and configuration_name
    like '%Tomcat Metrics%');
176
177 # Inserindo o Fato do CBO na Configuracao Open JDK8 Metrics ;
178 INSERT INTO 'F_Project_Metric'
179 ('percentil_value', 'D_Project_idProject', 'D_Metric_idMetric', '
    D_Quality_idQuality',
180 'D_Configuration_idConfiguration', 'D_Release_idRelease', 'D_Time_idTime
    ')
181 VALUES
182 (?, @idProject, @idCBO, @qualityBestCBO, @Best_Configuration, @idRelease
    , @idTime);
183
184 # Inserindo o Fato do CBO na Configuracao Tomcat Metrics ;
185 INSERT INTO 'F_Project_Metric'
186 ('percentil_value', 'D_Project_idProject', 'D_Metric_idMetric', '
    D_Quality_idQuality',
```

```
        D_Quality_idQuality',
187 'D_Configuration_idConfiguration', 'D_Release_idRelease', 'D_Time_idTime
        ')
188 VALUES
189 (?, @idProject, @idCBO, @qualityWorstCBO, @Worst_Configuration,
        @idRelease, @idTime);
190
191 # Consulta da Metrica LCOM4
192 SET @idDIT = (SELECT idMetric FROM D_Metric where metric_abbreviation='
        DIT');
193
194 # Consulta de Indicador de Qualidade de DIT na Configuracao Open JDK8
        Metrics
195 SET @qualityBestDIT = (SELECT idQuality FROM Meta_Metric_Ranges
196 INNER JOIN D_Quality
197 ON Meta_Metric_Ranges.quality_index=D_Quality.quality_index where
        language_name = @Project_Language
198 and metric_name='DIT' and ? <= max AND ? >= min and configuration_name
        like '%Open JDK8 Metrics%');
199
200 # Consulta de Indicador de Qualidade de DIT na Configuracao Tomcat
        Metrics
201 SET @qualityWorstDIT = (SELECT idQuality FROM Meta_Metric_Ranges
202 INNER JOIN D_Quality
203 ON Meta_Metric_Ranges.quality_index=D_Quality.quality_index where
        language_name = @Project_Language
204 and metric_name='DIT' and ? <= max AND ? >= min and configuration_name
        like '%Tomcat Metrics%');
205
206 # Inserindo o Fato do DIT na Configuracao Open JDK8 Metrics ;
207 INSERT INTO 'F_Project_Metric'
208 ('percentil_value', 'D_Project_idProject', 'D_Metric_idMetric', '
        D_Quality_idQuality',
209 'D_Configuration_idConfiguration', 'D_Release_idRelease', 'D_Time_idTime
        ')
210 VALUES
211 (?, @idProject, @idDIT, @qualityBestDIT, @Best_Configuration, @idRelease
        , @idTime);
212
213 # Inserindo o Fato do DIT na Configuracao Tomcat Metrics ;
214 INSERT INTO 'F_Project_Metric'
215 ('percentil_value', 'D_Project_idProject', 'D_Metric_idMetric', '
        D_Quality_idQuality',
216 'D_Configuration_idConfiguration', 'D_Release_idRelease', 'D_Time_idTime
        ')
217 VALUES
218 (?, @idProject, @idDIT, @qualityWorstDIT, @Worst_Configuration,
```

```
        @idRelease, @idTime);
219
220 # Consulta da Metrica LCOM4
221 SET @idLCOM4 = (SELECT idMetric FROM D_Metric where metric_abbreviation=
        'LCOM4');
222
223 # Consulta de Indicador de Qualidade de LCOM4 na Configuracao Open JDK8
        Metrics
224 SET @qualityBestLCOM4 = (SELECT idQuality FROM Meta_Metric_Ranges
225 INNER JOIN D_Quality
226 ON Meta_Metric_Ranges.quality_index=D_Quality.quality_index where
        language_name = @Project_Language
227 and metric_name='LCOM4' and ? <= max AND ? >= min and configuration_name
        like '%Open JDK8 Metrics%');
228
229 # Consulta de Indicador de Qualidade de LCOM4 na Configuracao Tomcat
        Metrics
230 SET @qualityWorstLCOM4 = (SELECT idQuality FROM Meta_Metric_Ranges
231 INNER JOIN D_Quality
232 ON Meta_Metric_Ranges.quality_index=D_Quality.quality_index where
        language_name = @Project_Language
233 and metric_name='LCOM4' and ? <= max AND ? >= min and configuration_name
        like '%Tomcat Metrics%');
234
235 # Inserindo o Fato do LCOM4 na Configuracao Open JDK8 Metrics ;
236 INSERT INTO 'F_Project_Metric '
237 ('percentil_value', 'D_Project_idProject', 'D_Metric_idMetric', '
        D_Quality_idQuality',
238 'D_Configuration_idConfiguration', 'D_Release_idRelease', 'D_Time_idTime
        ')
239 VALUES
240 (?, @idProject, @idLCOM4, @qualityBestLCOM4, @Best_Configuration,
        @idRelease, @idTime);
241
242 # Inserindo o Fato do LCOM4 na Configuracao Tomcat Metrics ;
243 INSERT INTO 'F_Project_Metric '
244 ('percentil_value', 'D_Project_idProject', 'D_Metric_idMetric', '
        D_Quality_idQuality',
245 'D_Configuration_idConfiguration', 'D_Release_idRelease', 'D_Time_idTime
        ')
246 VALUES
247 (?, @idProject, @idLCOM4, @qualityWorstLCOM4, @Worst_Configuration,
        @idRelease, @idTime);
248
249 # Consulta da Metrica NOC
250 SET @idNOC = (SELECT idMetric FROM D_Metric where metric_abbreviation='
        NOC');
```

```
251
252 # Consulta de Indicador de Qualidade de NOC na Configuracao Open JDK8
      Metrics
253 SET @qualityBestNOC = (SELECT idQuality FROM Meta_Metric_Ranges
254 INNER JOIN D_Quality
255 ON Meta_Metric_Ranges.quality_index=D_Quality.quality_index where
      language_name = @Project_Language
256 and metric_name='NOC' and ? <= max AND ? >= min and configuration_name
      like '%Open JDK8 Metrics%');
257
258 # Consulta de Indicador de Qualidade de NOC na Configuracao Tomcat
      Metrics
259 SET @qualityWorstNOC = (SELECT idQuality FROM Meta_Metric_Ranges
260 INNER JOIN D_Quality
261 ON Meta_Metric_Ranges.quality_index=D_Quality.quality_index where
      language_name = @Project_Language
262 and metric_name='NOC' and ? <= max AND ? >= min and configuration_name
      like '%Tomcat Metrics%');
263
264 # Inserindo o Fato do NOC na Configuracao Open JDK8 Metrics ;
265 INSERT INTO 'F_Project_Metric'
266 ('percentil_value', 'D_Project_idProject', 'D_Metric_idMetric', '
      D_Quality_idQuality',
267 'D_Configuration_idConfiguration', 'D_Release_idRelease', 'D_Time_idTime
      ')
268 VALUES
269 (?, @idProject, @idNOC, @qualityBestNOC, @Best_Configuration, @idRelease
      , @idTime);
270
271 # Inserindo o Fato do NOC na Configuracao Tomcat Metrics ;
272 INSERT INTO 'F_Project_Metric'
273 ('percentil_value', 'D_Project_idProject', 'D_Metric_idMetric', '
      D_Quality_idQuality',
274 'D_Configuration_idConfiguration', 'D_Release_idRelease', 'D_Time_idTime
      ')
275 VALUES
276 (?, @idProject, @idNOC, @qualityWorstNOC, @Worst_Configuration,
      @idRelease, @idTime);
277
278 # Consulta da Metrica NOM
279 SET @idNOM = (SELECT idMetric FROM D_Metric where metric_abbreviation='
      NOM');
280
281 # Consulta de Indicador de Qualidade de NOM na Configuracao Open JDK8
      Metrics
282 SET @qualityBestNOM = (SELECT idQuality FROM Meta_Metric_Ranges
283 INNER JOIN D_Quality
```

```
284 ON Meta_Metric_Ranges.quality_index=D_Quality.quality_index where
    language_name = @Project_Language
285 and metric_name='NOM' and ? <= max AND ? >= min and configuration_name
    like '%Open JDK8 Metrics%');
286
287 # Consulta de Indicador de Qualidade de NOM na Configuracao Tomcat
    Metrics
288 SET @qualityWorstNOM = (SELECT idQuality FROM Meta_Metric_Ranges
289 INNER JOIN D_Quality
290 ON Meta_Metric_Ranges.quality_index=D_Quality.quality_index where
    language_name = @Project_Language
291 and metric_name='NOM' and ? <= max AND ? >= min and configuration_name
    like '%Tomcat Metrics%');
292
293 # Inserindo o Fato do NOM na Configuracao Open JDK8 Metrics ;
294 INSERT INTO 'F_Project_Metric'
295 ('percentil_value', 'D_Project_idProject', 'D_Metric_idMetric', '
    D_Quality_idQuality',
296 'D_Configuration_idConfiguration', 'D_Release_idRelease', 'D_Time_idTime
    ')
297 VALUES
298 (?, @idProject, @idNOM, @qualityBestNOM, @Best_Configuration, @idRelease
    , @idTime);
299
300 # Inserindo o Fato do NOM na Configuracao Tomcat Metrics ;
301 INSERT INTO 'F_Project_Metric'
302 ('percentil_value', 'D_Project_idProject', 'D_Metric_idMetric', '
    D_Quality_idQuality',
303 'D_Configuration_idConfiguration', 'D_Release_idRelease', 'D_Time_idTime
    ')
304 VALUES
305 (?, @idProject, @idNOM, @qualityWorstNOM, @Worst_Configuration,
    @idRelease, @idTime);
306
307 # Consulta da Metrica NPA
308 SET @idNPA = (SELECT idMetric FROM D_Metric where metric_abbreviation='
    NPA');
309
310 # Consulta de Indicador de Qualidade de NPA na Configuracao Open JDK8
    Metrics
311 SET @qualityBestNPA = (SELECT idQuality FROM Meta_Metric_Ranges
312 INNER JOIN D_Quality
313 ON Meta_Metric_Ranges.quality_index=D_Quality.quality_index where
    language_name = @Project_Language
314 and metric_name='NPA' and ? <= max AND ? >= min and configuration_name
    like '%Open JDK8 Metrics%');
315
```

```
316 # Consulta de Indicador de Qualidade de NPA na Configuracao Tomcat
      Metrics
317 SET @qualityWorstNPA = (SELECT idQuality FROM Meta_Metric_Ranges
318 INNER JOIN D_Quality
319 ON Meta_Metric_Ranges.quality_index=D_Quality.quality_index where
      language_name = @Project_Language
320 and metric_name='NPA' and ? <= max AND ? >= min and configuration_name
      like '%Tomcat Metrics%');
321
322 # Inserindo o Fato do NPA na Configuracao Open JDK8 Metrics ;
323 INSERT INTO 'F_Project_Metric '
324 ('percentil_value', 'D_Project_idProject', 'D_Metric_idMetric', '
      D_Quality_idQuality',
325 'D_Configuration_idConfiguration', 'D_Release_idRelease', 'D_Time_idTime
      ')
326 VALUES
327 (?, @idProject, @idNPA, @qualityBestNPA, @Best_Configuration, @idRelease
      , @idTime);
328
329 # Inserindo o Fato do NPA na Configuracao Tomcat Metrics ;
330 INSERT INTO 'F_Project_Metric '
331 ('percentil_value', 'D_Project_idProject', 'D_Metric_idMetric', '
      D_Quality_idQuality',
332 'D_Configuration_idConfiguration', 'D_Release_idRelease', 'D_Time_idTime
      ')
333 VALUES
334 (?, @idProject, @idNPA, @qualityWorstNPA, @Worst_Configuration,
      @idRelease, @idTime);
335
336 # Consulta da Metrica RFC
337 SET @idRFC = (SELECT idMetric FROM D_Metric where metric_abbreviation='
      RFC');
338
339 # Consulta de Indicador de Qualidade de RFC na Configuracao Open JDK8
      Metrics
340 SET @qualityBestRFC = (SELECT idQuality FROM Meta_Metric_Ranges
341 INNER JOIN D_Quality
342 ON Meta_Metric_Ranges.quality_index=D_Quality.quality_index where
      language_name = @Project_Language
343 and metric_name='RFC' and ? <= max AND ? >= min and configuration_name
      like '%Open JDK8 Metrics%');
344
345 # Consulta de Indicador de Qualidade de RFC na Configuracao Tomcat
      Metrics
346 SET @qualityWorstRFC = (SELECT idQuality FROM Meta_Metric_Ranges
347 INNER JOIN D_Quality
348 ON Meta_Metric_Ranges.quality_index=D_Quality.quality_index where
```

```

    language_name = @Project_Language
349 and metric_name='RFC' and ? <= max AND ? >= min and configuration_name
    like '%Tomcat Metrics%');
350
351 # Inserindo o Fato do RFC na Configuracao Open JDK8 Metrics ;
352 INSERT INTO 'F_Project_Metric'
353 ('percentil_value', 'D_Project_idProject', 'D_Metric_idMetric', '
    D_Quality_idQuality',
354 'D_Configuration_idConfiguration', 'D_Release_idRelease', 'D_Time_idTime
    ')
355 VALUES
356 (?, @idProject, @idRFC, @qualityBestRFC, @Best_Configuration, @idRelease
    , @idTime);
357
358 # Inserindo o Fato do RFC na Configuracao Tomcat Metrics ;
359 INSERT INTO 'F_Project_Metric'
360 ('percentil_value', 'D_Project_idProject', 'D_Metric_idMetric', '
    D_Quality_idQuality',
361 'D_Configuration_idConfiguration', 'D_Release_idRelease', 'D_Time_idTime
    ')
362 VALUES
363 (?, @idProject, @idRFC, @qualityWorstRFC, @Worst_Configuration,
    @idRelease, @idTime);

```

Código-Fonte 3 – Script SQL de Avaliação dos Valores Percentis das Métricas de Código-Fonte

Na terceira transformação, como se mostra na Figura 30, foram cobertos os processos de negócio de avaliar os cenários de limpeza de código-fonte em cada classe do Projeto em uma determinada *release* e o cálculo da Taxa de Aproveitamento de Oportunidades de Melhoria de Código-Fonte. Para tal, foram obtidos os valores cada métrica para cada classe utilizando o componente *JSON Input*. Após a coleta das métricas, foram inseridas, utilizando o componente *Combination Lookup* a fim de evitar duplicações ao longo das *releases*, as classes do projeto.



Figura 30 – Terceira Transformação realizada no Kettle

Por fim, foram identificados os Cenários de Limpeza de Código-Fonte com auxílio dos metadados utilizando o componente *Execute SQL Script*. Aind neste componente, foi

calculada a Taxa de Aproveitamento de Oportunidades de Melhoria de Código-Fonte a partir a soma de todos cenários de limpeza identificados em uma determinada release e a soma de todas as classes. O código-fonte, que foi colocado no componente *Execute SQL Script* é descrito no Código-Fonte 4, onde cada foi substituído por uma variável dentro da *Transformation*.

```
1 use source_info;
2
3 #Obtendo o numero da Release
4 SET @release_number = (SELECT substring_index('?', '.', 1));
5
6 #Obtendo o id da Release
7 SET @idRelease = (SELECT idRelease from D_Release where release_number =
      @release_number);
8
9 #Obtendo a linguagem de programacao do Projeto
10 SET @Project_Language = (SELECT project_language from D_Project);
11
12 #Obtendo o Id do Projeto
13 SET @idProject = (SELECT max(idProject) from D_Project);
14
15 #Verificando no Metadados a Range do ACCM
16 SET @idACCM = (SELECT idMetricRange FROM Meta_Metric_Ranges
17 where language_name = @Project_Language
18 and metric_name='ACCM' and ? <= max AND ? >= min and configuration_name
      like '%Open JDK8 Metrics%');
19
20 #Verificando no Metadados a Range do AMLOC
21 SET @idAMLOC = (SELECT idMetricRange FROM Meta_Metric_Ranges
22 where language_name = @Project_Language
23 and metric_name='AMLOC' and ? <= max AND ? >= min and configuration_name
      like '%Open JDK8 Metrics%');
24
25 #Verificando no Metadados a Range do ANPM
26 SET @idANPM = (SELECT idMetricRange FROM Meta_Metric_Ranges
27 where language_name = @Project_Language
28 and metric_name='ANPM' and ? <= max AND ? >= min and configuration_name
      like '%Open JDK8 Metrics%');
29
30 #Verificando no Metadados a Range do CBO
31 SET @idCBO = (SELECT idMetricRange FROM Meta_Metric_Ranges
32 where language_name = @Project_Language
33 and metric_name='CBO' and ? <= max AND ? >= min and configuration_name
      like '%Open JDK8 Metrics%');
34
35 #Verificando no Metadados a Range do LCOM4
36 SET @idLCOM4 = (SELECT idMetricRange FROM Meta_Metric_Ranges
```

```
37 where language_name = @Project_Language
38 and metric_name='LCOM4' and ? <= max AND ? >= min and configuration_name
    like '%Open JDK8 Metrics%');
39
40 #Verificando no Metadados a Range do NPA
41 SET @idNPA = (SELECT idMetricRange FROM Meta_Metric_Ranges
42 where language_name = @Project_Language
43 and metric_name='NPA' and ? <= max AND ? >= min and configuration_name
    like '%Open JDK8 Metrics%');
44
45 #Verificando no Metadados a Range do NOC
46 SET @idNOC = (SELECT idMetricRange FROM Meta_Metric_Ranges
47 where language_name = @Project_Language
48 and metric_name='NOC' and ? <= max AND ? >= min and configuration_name
    like '%Open JDK8 Metrics%');
49
50 #Verificando no Metadados a Range do RFC
51 SET @idRFC = (SELECT idMetricRange FROM Meta_Metric_Ranges
52 where language_name = @Project_Language
53 and metric_name='RFC' and ? <= max AND ? >= min and configuration_name
    like '%Open JDK8 Metrics%');
54
55 #Verificando a existencia de Classe Pouco Coesa
56 SET @idClassePoucoCoesa = (SELECT 'Meta_Scenario'. 'idMeta_Scenario'
57 FROM 'Meta_Metric_Ranges_Meta_Scenario' INNER JOIN Meta_Scenario ON
    Meta_Scenario.idMeta_Scenario = Meta_Metric_Ranges_Meta_Scenario.
    Meta_Scenario_idMeta_Scenario where
58 Meta_Metric_Ranges_Meta_Scenario.Meta_Metric_Ranges_idMetricRange1 =
    @idLCOM4
59 and Meta_Metric_Ranges_Meta_Scenario.Meta_Metric_Ranges_idMetricRange2
    = @idRFC);
60
61 #Verificando a existencia de Interface dos Metodos
62 SET @idInterfaceMetodos = (SELECT 'Meta_Scenario'. 'idMeta_Scenario'
63 FROM 'Meta_Metric_Ranges_Meta_Scenario' INNER JOIN Meta_Scenario ON
    Meta_Scenario.idMeta_Scenario = Meta_Metric_Ranges_Meta_Scenario.
    Meta_Scenario_idMeta_Scenario where
64 Meta_Metric_Ranges_Meta_Scenario.Meta_Metric_Ranges_idMetricRange1 =
    @idANPM);
65
66 #Verificando a existencia de Classe com muitos Filhos
67 SET @idClasseFilhos = (SELECT 'Meta_Scenario'. 'idMeta_Scenario'
68 FROM 'Meta_Metric_Ranges_Meta_Scenario' INNER JOIN Meta_Scenario ON
    Meta_Scenario.idMeta_Scenario = Meta_Metric_Ranges_Meta_Scenario.
    Meta_Scenario_idMeta_Scenario where
69 Meta_Metric_Ranges_Meta_Scenario.Meta_Metric_Ranges_idMetricRange1 =
    @idNOC);
```

```
70
71 #Verificando a existencia de Classe Grande
72 SET @idClasseGrande = (SELECT 'Meta_Scenario'.'idMeta_Scenario'
73 FROM 'Meta_Metric_Ranges_Meta_Scenario' INNER JOIN Meta_Scenario ON
    Meta_Scenario.idMeta_Scenario = Meta_Metric_Ranges_Meta_Scenario.
    Meta_Scenario_idMeta_Scenario where
74 Meta_Metric_Ranges_Meta_Scenario.Meta_Metric_Ranges_idMetricRange1 =
    @idACCM
75 and Meta_Metric_Ranges_Meta_Scenario.Meta_Metric_Ranges_idMetricRange2 =
    @idAMLOC);
76
77 #Verificando a Existencia de Classe Exposta
78 SET @idClasseExposta = (SELECT 'Meta_Scenario'.'idMeta_Scenario'
79 FROM 'Meta_Metric_Ranges_Meta_Scenario' INNER JOIN Meta_Scenario ON
    Meta_Scenario.idMeta_Scenario = Meta_Metric_Ranges_Meta_Scenario.
    Meta_Scenario_idMeta_Scenario where
80 Meta_Metric_Ranges_Meta_Scenario.Meta_Metric_Ranges_idMetricRange1 =
    @idNPA);
81
82 #Verificando a Existencia de Complexidade Estrutural
83 SET @idComplexidadeEstrutural = (SELECT 'Meta_Scenario'.'idMeta_Scenario
84 FROM 'Meta_Metric_Ranges_Meta_Scenario' INNER JOIN Meta_Scenario ON
    Meta_Scenario.idMeta_Scenario = Meta_Metric_Ranges_Meta_Scenario.
    Meta_Scenario_idMeta_Scenario where
85 Meta_Metric_Ranges_Meta_Scenario.Meta_Metric_Ranges_idMetricRange1 =
    @idCBO
86 and Meta_Metric_Ranges_Meta_Scenario.Meta_Metric_Ranges_idMetricRange2
    = @idLCOM4);
87
88 #Criando uma Tabela Temporaria para receber os Cenarios
89 DROP TABLE IF EXISTS Temporary_F_Scenario_Class ;
90 CREATE Temporary TABLE 'source_info'.'Temporary_F_Scenario_Class' (
91 'idScenarioFact' INT NOT NULL AUTO_INCREMENT,
92 'quantity_Scenario' INT NULL,
93 'D_Scenario_Clean_Code_idScenario' INT NULL,
94 'D_Project_idProject' INT NULL,
95 'D_Release_idRelease' INT NULL,
96 'D_Class_idClass' INT NULL,
97 PRIMARY KEY ('idScenarioFact'))
98 ENGINE = InnoDB;
99
100 #Inserindo o Fato de Classe Pouco Coesa na Tabela Temporaria
101 INSERT INTO 'source_info'.'Temporary_F_Scenario_Class'
102 ('quantity_Scenario','D_Scenario_Clean_Code_idScenario',
103 'D_Project_idProject',
104 'D_Release_idRelease',
```

```
105 'D_Class_idClass '
106 VALUES
107 (1, @idClassePoucoCoesa, @idProject, @idRelease, ?);
108
109 #Inserindo o Fato de Interface dos Metodos na Tabela Temporaria
110
111 INSERT INTO 'source_info'. 'Temporary_F_Scenario_Class '
112 ('quantity_Scenario', 'D_Scenario_Clean_Code_idScenario',
113 'D_Project_idProject',
114 'D_Release_idRelease',
115 'D_Class_idClass ')
116 VALUES
117 (1, @idInterfaceMetodos, @idProject, @idRelease, ?);
118
119
120 #Inserindo o Fato Classe com Muitos Filhos na Tabela Temporaria
121 INSERT INTO 'source_info'. 'Temporary_F_Scenario_Class '
122 ('quantity_Scenario', 'D_Scenario_Clean_Code_idScenario',
123 'D_Project_idProject',
124 'D_Release_idRelease',
125 'D_Class_idClass ')
126 VALUES
127 (1, @idClasseFilhos, @idProject, @idRelease, ?);
128
129
130 #Inserindo o Fato da Classe com Metodos Grandes ou muitos condicionais
    na Tabela Temporaria
131 INSERT INTO 'source_info'. 'Temporary_F_Scenario_Class '
132 ('quantity_Scenario', 'D_Scenario_Clean_Code_idScenario',
133 'D_Project_idProject',
134 'D_Release_idRelease',
135 'D_Class_idClass ')
136 VALUES
137 (1, @idClasseGrande, @idProject, @idRelease, ?);
138
139
140 #Inserindo o Fato da Classe muito exposta na Tabela Temporaria
141 INSERT INTO 'source_info'. 'Temporary_F_Scenario_Class '
142 ('quantity_Scenario', 'D_Scenario_Clean_Code_idScenario',
143 'D_Project_idProject',
144 'D_Release_idRelease',
145 'D_Class_idClass ')
146 VALUES
147 (1, @idClasseExposta, @idProject, @idRelease, ?);
148
149 #Inserindo o Fato da Classe com grande complexidade estrutural na Tabela
    Temporaria
```

```
150 INSERT INTO `source_info`.`Temporary_F_Scenario_Class`
151 (`quantity_Scenario`,`D_Scenario_Clean_Code_idScenariio`,`
152 `D_Project_idProject`,`
153 `D_Release_idRelease`,`
154 `D_Class_idClass`)
155 VALUES
156 (1, @idComplexidadeEstrutural, @idProject, @idRelease, ?);
157
158
159 #Copiando da Tabela Temporaria para Tabela Fato apenas os que nao sao
      nulos
160 INSERT INTO F_Scenario_Class (`F_Scenario_Class`.`quantity_Scenario`,`
      F_Scenario_Class`.`D_Scenario_Clean_Code_idScenariio`,`
161 `F_Scenario_Class`.`D_Project_idProject`,`
162 `F_Scenario_Class`.`D_Release_idRelease`,`
163 `F_Scenario_Class`.`D_Class_idClass`) SELECT `Temporary_F_Scenario_Class`
      `.`quantity_Scenario`,`Temporary_F_Scenario_Class`.`
      D_Scenario_Clean_Code_idScenariio`,`
164 `Temporary_F_Scenario_Class`.`D_Project_idProject`,`
165 `Temporary_F_Scenario_Class`.`D_Release_idRelease`,`
166 `Temporary_F_Scenario_Class`.`D_Class_idClass` FROM
      Temporary_F_Scenario_Class where Temporary_F_Scenario_Class.
      D_Scenario_Clean_Code_idScenariio is not null;
167
168 #Calculando o total de cenarios na release
169 SET @total_scenarios = (SELECT COUNT(*)FROM source_info.F_Scenario_Class
      where D_Release_idRelease= @idRelease);
170
171 #Calculando a Taxa de Aproveitamento de Oportunidades de Melhoria de
      Codigo-Fonte
172 SET @rate = (@total_scenarios/?);
173
174 #Criando uma Tabela Temporaria que recebera a Taxa, numero de Classes e
      Quantidade de Cenarios
175 DROP TABLE IF EXISTS `Temporary_F_Rate_Scenario`;
176 CREATE Temporary TABLE `Temporary_F_Rate_Scenario` (
177   `idRateScenariio` int(11) NOT NULL AUTO_INCREMENT,
178   `RateScenariio` double DEFAULT NULL,
179   `Quantiy_Scenarios` double DEFAULT NULL,
180   `numberOfClasses` int(11) DEFAULT NULL,
181   `D_Project_idProject` int(11) NOT NULL,
182   `D_Release_idRelease` int(11) NOT NULL,
183   PRIMARY KEY (`idRateScenariio`))
184 ENGINE=InnoDB DEFAULT CHARSET=utf8;
185
186 #Inserindo na Tabela Temporaria a Taxa, numero de Classes e Quantidade
      de Cenarios
```

```
187 INSERT INTO 'source_info'. 'Temporary_F_Rate_Scenario '  
188 ('RateScenario', 'Quantiy_Scenarios', 'numberOfClasses',  
189 'D_Project_idProject', 'D_Release_idRelease')  
190 VALUES  
191 (@rate, @total_scenarios, ?, @idProject, @idRelease);  
192  
193 #Passando os dados da Tabela Temporaria para Tabela Fato  
194 REPLACE INTO 'source_info'. 'F_Rate_Scenario '  
195 SET 'RateScenario' = @rate, 'numberOfClasses' = ?, 'Quantiy_Scenarios' =  
    @total_scenarios,  
196 'D_Release_idRelease' = @idRelease, 'D_Project_idProject' = @idProject;
```

Código-Fonte 4 – *Script* SQL de Identificação de Cenários de Limpeza de Código-Fonte e Cálculo da Taxa de Aproveitamento de Oportunidades de Melhoria de Código-Fonte

A.2 Implementação do *Job*

Como explicado anteriormente, o Kettle utiliza o *Job* para executar tarefas, em nível mais alto, de fluxo de controle, tais como, mandar um email em caso de falha, baixar um arquivo, executar transformações e entre outras atividades. Dessa forma os principais componentes internos do *Job*, que foram utilizados no trabalho, são mostrados na Figura 31.



Figura 31 – Componentes do Kettle que foram utilizadas nos *Jobs*

O componente *Start* é utilizado para marcar o início da execução de um determinado *Job*. Nele, é possível programar a execução repetida de um determinado *job*, como por exemplo, a cada hora, dia ou mês; Já o componente *Transformation* serve para executar uma determinada Transformação, que fora especificada anteriormente.

Após se contruir o arquivo *Job* do presente trabalho, obteve-se a Figura 32.



Figura 32 – Job deste Trabalho

No *Job*, a transformação "Dados do Projeto" corresponde a execução da Figura 28. Já a transformação "Análise dos Valores Percentis" corresponde a Figura 29 e por fim, a transformação "Análise dos Cenários de Limpeza de Código-Fonte" corresponde a Figura 30.

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

Neste apêndice, serão apresentados os gráficos e tabelas, obtidos com o plugin Saiku no ambiente de *Data Warehousing*, dos Percentis obtidos para cada uma das métricas de código-fonte no Sistema Integrado de Conhecimento e Gestão (SICG) do Instituto do Patrimônio Arstítico Nacional (IPHAN).

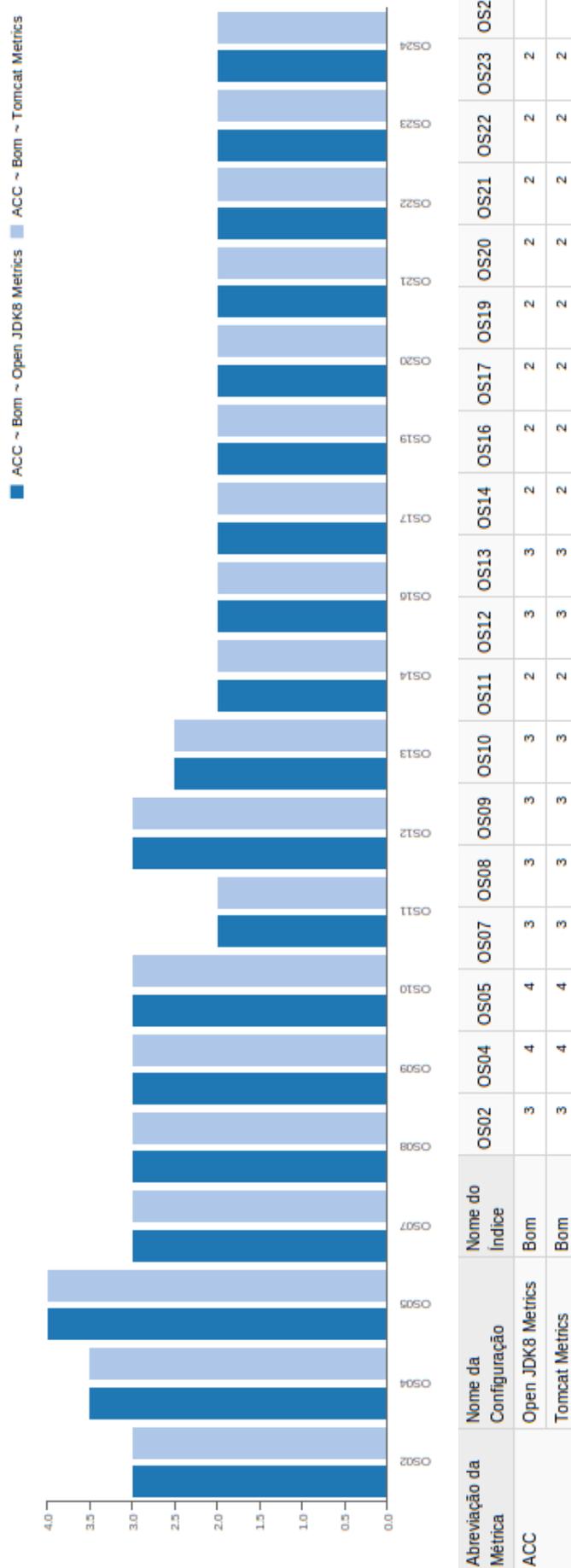


Figura 33 – Intepretação dos Valores Percentis da Métrica ACC

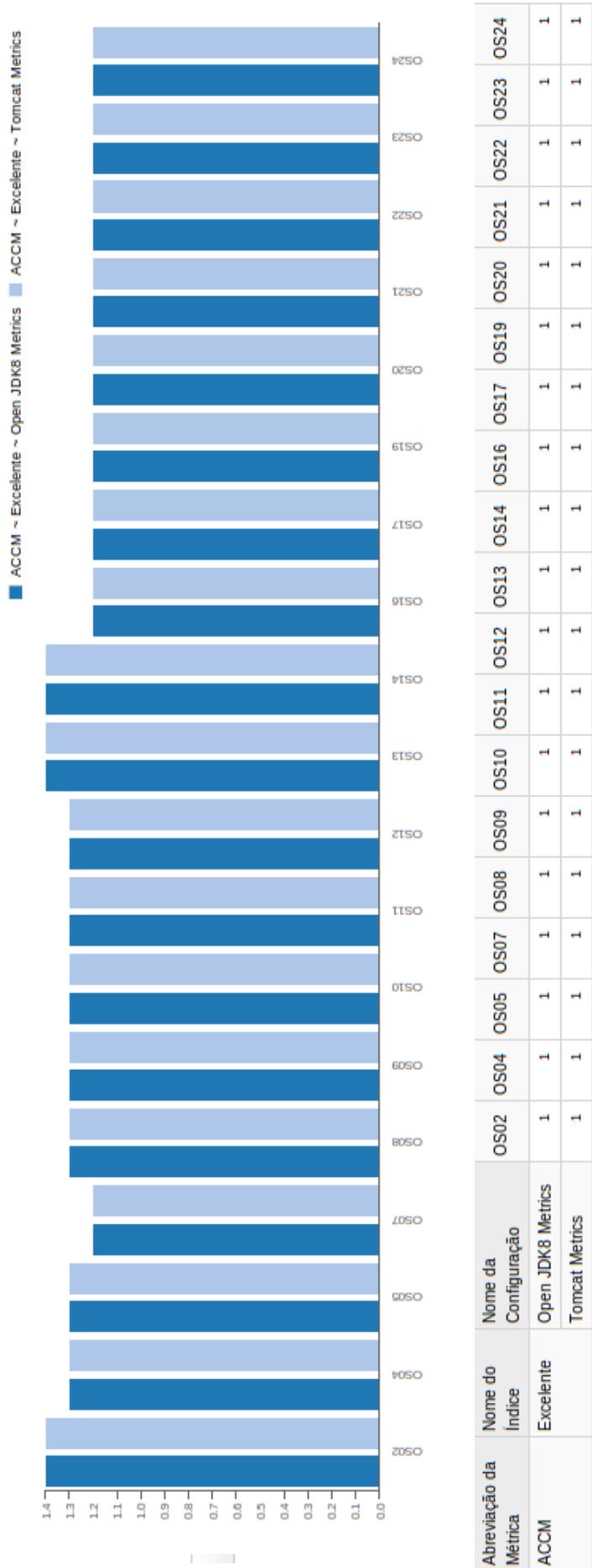


Figura 34 – Intepretação dos Valores Percentis da Métrica ACCM



Figura 35 – Intepretação dos Valores Percentis da Métrica AMLOC



Figura 36 – Interpretação dos Valores Percentis da Métrica ANPM

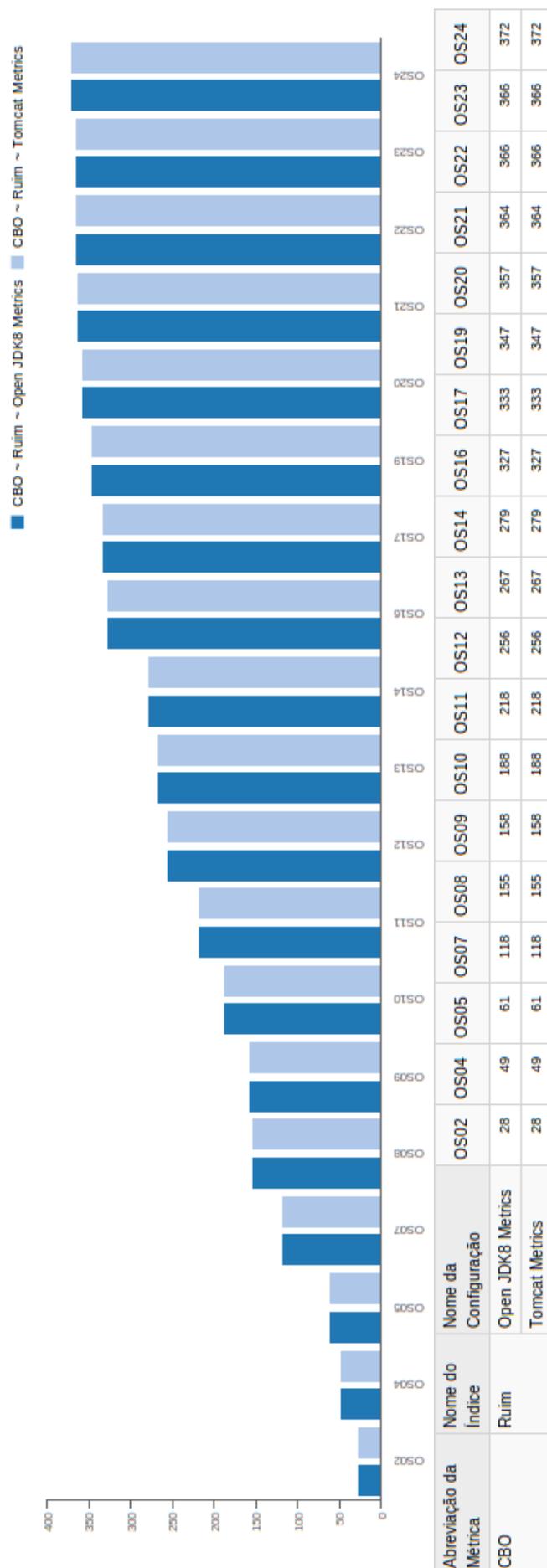


Figura 37 – Intepretação dos Valores Percentis da Métrica CBO



Figura 38 – Intepretação dos Valores Percentis da Métrica DIT



Figura 39 – Interpretação dos Valores Percentis da Métrica LCOM4

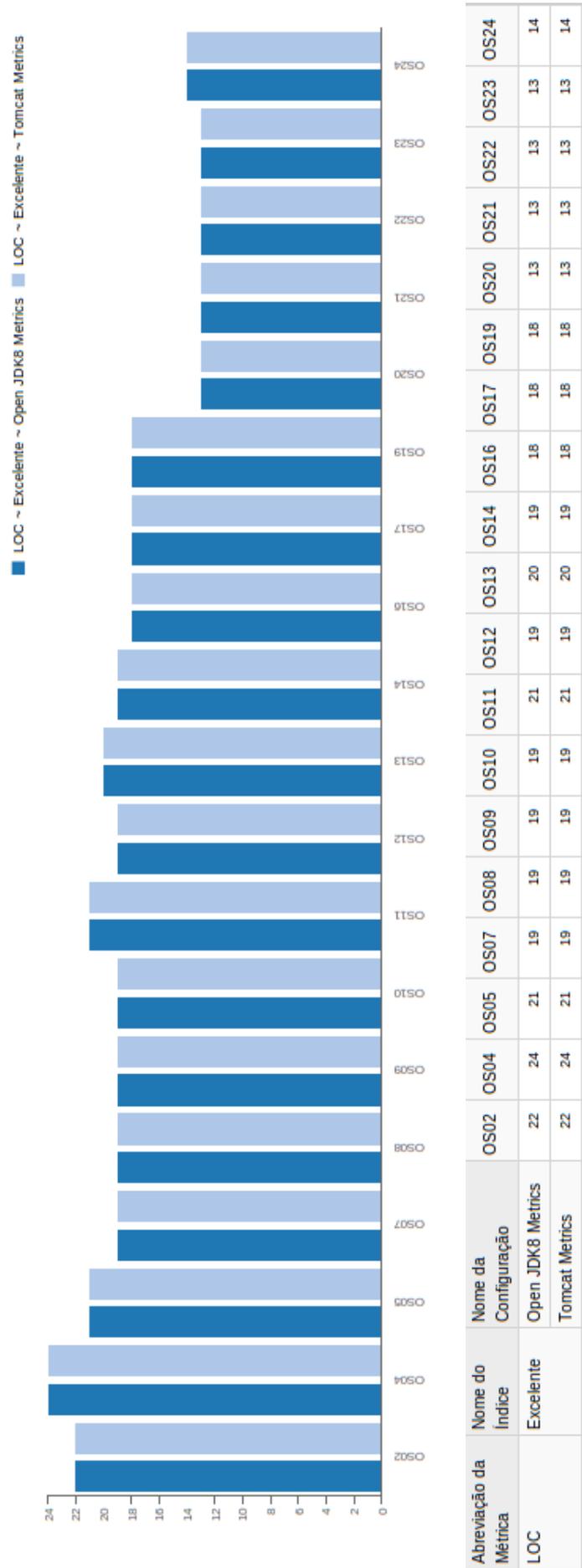


Figura 40 – Intepretação dos Valores Percentis da Métrica LOC

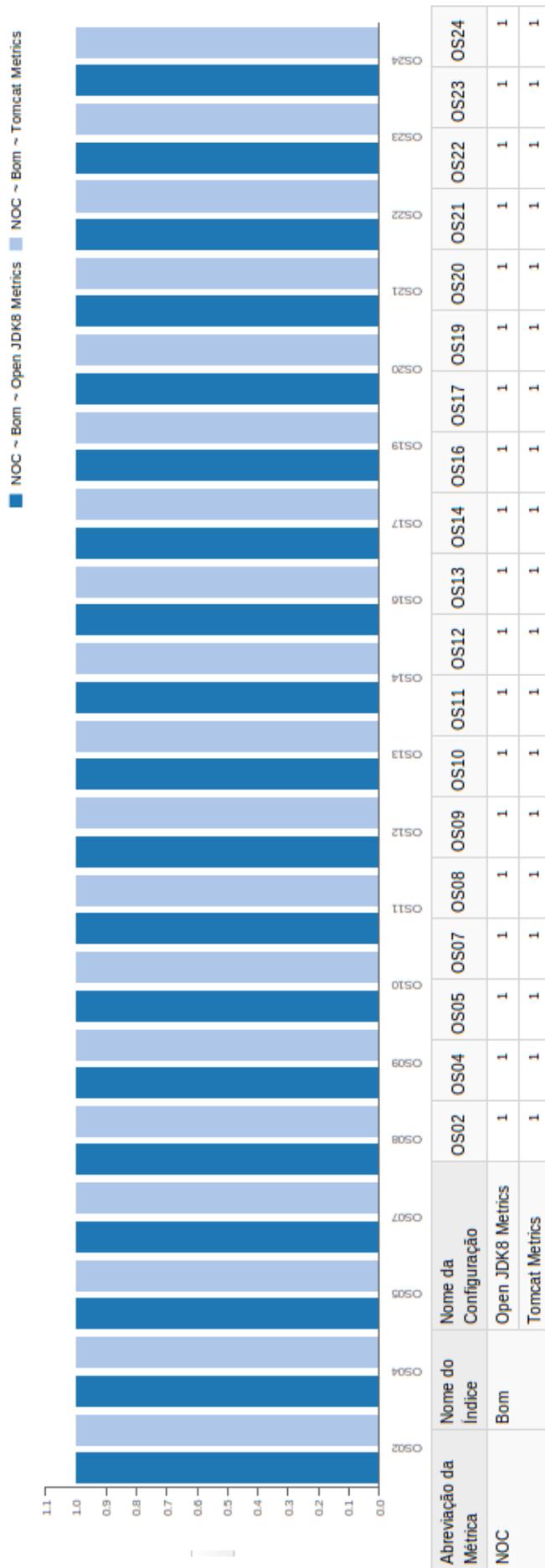


Figura 41 – Intepretação dos Valores Percentis da Métrica NOC



Figura 42 – Intepretação dos Valores Percentis da Métrica NOM

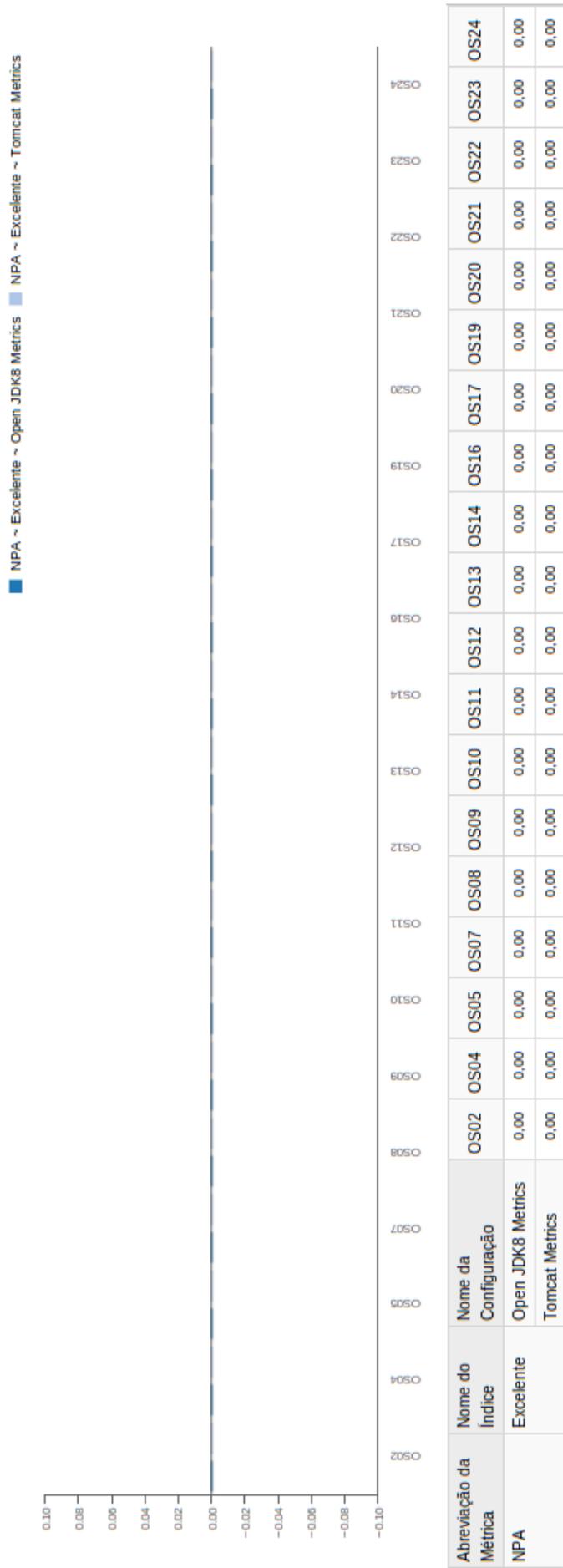


Figura 43 – Intepretação dos Valores Percentis da Métrica NPA



Figura 44 – Intepretação dos Valores Percentis da Métrica RFC

APÊNDICE C – Cenários de Limpeza de Código-Fonte

Neste apêndice, são detalhadas as classes e os cenários de limpeza de código-fonte que foram identificados para as mesmas ao longo das *releases* do projeto.

Nome da Classe	Nome do Cenário	OS02	OS04	OS07	OS08	OS09	OS10	OS11	OS12	OS13	OS14	OS16	OS17	OS19	OS20	OS21	OS22	OS23	OS24
br.gov.iphan.sig.apresentacao.controllers.AcaoController	Classe com métodos grandes e/ou muitos condic									1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
br.gov.iphan.sig.apresentacao.controllers.BemController	Classe com métodos grandes e/ou muitos condic												1.0			1.0	1.0		
br.gov.iphan.sig.apresentacao.controllers:BemImaterialAcaoController	Classe com métodos grandes e/ou muitos condic									1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
br.gov.iphan.sig.apresentacao.controllers:BemImaterialController	Classe com métodos grandes e/ou muitos condic										1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
br.gov.iphan.sig.apresentacao.controllers:BemImovelCaracterizacaoExternaController	Classe com métodos grandes e/ou muitos condic				1.0														
br.gov.iphan.sig.apresentacao.controllers:BemImovelCaracterizacaoInternaController	Classe com métodos grandes e/ou muitos condic				1.0	1.0	1.0							1.0	1.0	1.0	1.0	1.0	1.0
br.gov.iphan.sig.apresentacao.controllers:BemImovelConjuntosController	Classe com métodos grandes e/ou muitos condic					1.0	1.0	1.0						1.0	1.0				
br.gov.iphan.sig.apresentacao.controllers:BemImovelLoteController	Classe com métodos grandes e/ou muitos condic												1.0	1.0	1.0	1.0	1.0	1.0	1.0
br.gov.iphan.sig.apresentacao.controllers:BemProtecaoController	Classe com métodos grandes e/ou muitos condic									1.0	1.0	1.0							
br.gov.iphan.sig.apresentacao.controllers:ClassificacaoTaxonomicaMaterialPreHistoricoController	Classe com métodos grandes e/ou muitos condic								1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
br.gov.iphan.sig.apresentacao.controllers:ColecaoController	Classe com métodos grandes e/ou muitos condic											1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
br.gov.iphan.sig.apresentacao.controllers:FormacaoController	Classe com métodos grandes e/ou muitos condic			1.0	1.0	1.0													
br.gov.iphan.sig.apresentacao.controllers:FiscalizacaoWsController	Classe com métodos grandes e/ou muitos condic							1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
br.gov.iphan.sig.apresentacao.controllers:HomologacaoController	Classe com métodos grandes e/ou muitos condic																		1.0
br.gov.iphan.sig.apresentacao.controllers:InstituicaoVincularBemImaterialController	Classe com métodos grandes e/ou muitos condic															1.0	1.0	1.0	1.0
br.gov.iphan.sig.apresentacao.controllers:PesquisaBemController	Classe com métodos grandes e/ou muitos condic										1.0								
br.gov.iphan.sig.apresentacao.controllers:PreSetorController	Classe com métodos grandes e/ou muitos condic									1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
br.gov.iphan.sig.apresentacao.controllers:PreSetorizacaoController	Classe com métodos grandes e/ou muitos condic									1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
br.gov.iphan.sig.apresentacao.controllers:ReferenciaBibliograficaController	Classe com métodos grandes e/ou muitos condic	1.0																	
br.gov.iphan.sig.apresentacao.controllers:RelatorioBemImaterial	Classe com métodos grandes e/ou muitos condic																		1.0
br.gov.iphan.sig.apresentacao.controllers:SetorizacaoController	Classe com métodos grandes e/ou muitos condic												1.0	1.0		1.0	1.0	1.0	1.0
br.gov.iphan.sig.apresentacao.controllers:SitioPreservacaoController	Classe com métodos grandes e/ou muitos condic								1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
br.gov.iphan.sig.apresentacao.controllers:VisualizarController	Classe com métodos grandes e/ou muitos condic									1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
br.gov.iphan.sig.apresentacao.converters:FormaAcondicionamentoConverter	Classe com métodos grandes e/ou muitos condic											1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
br.gov.iphan.sig.apresentacao.converters:FormaAquisicaoConverter	Classe com métodos grandes e/ou muitos condic											1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
br.gov.iphan.sig.apresentacao.converters:GeometryConverter	Classe com métodos grandes e/ou muitos condic													1.0	1.0	1.0	1.0	1.0	1.0
br.gov.iphan.sig.apresentacao.util:VinculacaoMultimediaUtil	Classe com métodos grandes e/ou muitos condic							1.0	1.0	1.0	1.0	1.0	1.0						
br.gov.iphan.sig.comun.util:PesquisarAcaoStrategy	Classe com métodos grandes e/ou muitos condic													1.0	1.0				
br.gov.iphan.sig.comun.util:PesquisarBemStrategyContext	Classe com métodos grandes e/ou muitos condic										1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
br.gov.iphan.sig.comun.util:ShapelleUtil	Classe com métodos grandes e/ou muitos condic							1.0	1.0	1.0	1.0	1.0	1.0						
br.gov.iphan.sig.comun.util:ShapelleZibStreamReader	Classe com métodos grandes e/ou muitos condic													1.0	1.0	1.0	1.0	1.0	1.0
br.gov.iphan.sig.negocio:util:PesquisaVehiculo	Classe com métodos grandes e/ou muitos condic													1.0	1.0	1.0	1.0	1.0	1.0
br.gov.iphan.sig.persistencia:dao:impl:BemDAOImpl	Classe com métodos grandes e/ou muitos condic								1.0	1.0	1.0	1.0		1.0					
br.gov.iphan.sig.persistencia:dao:impl:BemImaterialDAOImpl	Classe com métodos grandes e/ou muitos condic															1.0	1.0		
br.gov.iphan.sig.persistencia:dao:impl:PesquisaBemDAOImpl	Classe com métodos grandes e/ou muitos condic																		1.0
br.gov.iphan.sig.persistencia:dao:impl:ReferenciaBibliograficaDAOImpl	Classe com métodos grandes e/ou muitos condic	1.0																	

Tabela 33 – Classes com Cenário de Limpeza: Classe com métodos muito grandes e/ou muitos condicionais

Nome da Classe	Nome do Cenário	OS02	OS04	OS05	OS07	OS08	OS09	OS10	OS11	OS12	OS13	OS14	OS16	OS17	OS19	OS20	OS21	OS22	OS23	OS24
br.gov.iphan.sig.comun.util:AbstractPesquisarBemStrategy	Classes com muitos filhos										1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
br.gov.iphan.sig.comun.util:VincularMultimediaStrategy	Classes com muitos filhos								1.0	1.0										1.0
br.gov.iphan.sig.negocio:modelo:FormaAcondicionamento	Classes com muitos filhos											1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
br.gov.iphan.sig.negocio:modelo:FormaAquisicao	Classes com muitos filhos											1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
br.gov.iphan.sig.negocio:modelo:ReferenciaBibliografica	Classes com muitos filhos	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0

Tabela 34 – Classes com Cenário de Limpeza: Classe com muitos filhos

Nome da Classe	Nome do Cenário	OS13	OS14	OS16	OS17	OS19	OS20	OS21	OS22	OS23	OS24
br.gov.iphan.sig.apresentacao.controllers:relatorios:RelatorioBemController:Line	Classe com muita Exposição									1.0	1.0
br.gov.iphan.sig.comun.util:AbstractPesquisarBemStrategy	Classe com muita Exposição	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0

Tabela 35 – Classes com Cenário de Limpeza: Classe com muita exposição

