



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*

Autor: Guilherme Baufaker Rêgo
Orientador: Prof. Msc. Hilmer Rodrigues Neri
Coorientador: Prof. Dr. Paulo Roberto Miranda Meirelles

Brasília, DF
2014



Guilherme Baufaker Rêgo

Monitoramento de Métricas de Código-Fonte com suporte de um ambiente de *Data Warehousing*

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

Coorientador: Prof. Dr. Paulo Roberto Miranda Meirelles

Brasília, DF

2014

Guilherme Baufaker Rêgo

Monitoramento de Métricas de Código-Fonte com suporte de um ambiente de
Data Warehousing/ Guilherme Baufaker Rêgo. – Brasília, DF, 2014-
70 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Msc. Hilmer Rodrigues Neri

Coorientador: Prof. Dr. Paulo Roberto Miranda Meirelles

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

CDU 02:141:005.6

Guilherme Baufaker Rêgo

Monitoramento de Métricas de Código-Fonte com suporte de um ambiente de *Data Warehousing*

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, 06 de Julho de 2014:

Prof. Msc. Hilmer Rodrigues Neri
Orientador

**Prof. Dr. Paulo Roberto Miranda
Meirelles**
Coorientador

**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, desde pelas oportunidades concedidas no projeto Desafio Positivo e no SRA, quanto a orientação no presente trabalho.

Agradeço também ao meu co-orientador, Prof. Paulo Meirelles por ter me mostrado o mundo do software livre, por ter compartilhado conhecimento sobre Métricas de Código-Fonte, Git, Ruby, Rails e em especial por ter compartilhado suas ponderações muito importantes para o meu crescimento pessoal e profissional.

Agradeço a Deus, inteligência suprema criadora de todas as coisas, por cada dia que é uma nova chance no caminho da evolução. e 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 minha maior parte da criação, devo a ela desde todo o suporte fornecido em casa até a preocupação sobre a quantidade de horas de sono estava dormindo durante a noite.

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

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

Resumo a Fazer

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

Abstract

To do the ABstract

Palavras-chaves: *Source Code Metrics. Data Warehousing. Data Warehouse*

Lista de ilustrações

Figura 1 – Modelo de Informação da ISO 15939	20
Figura 2 – Modelo de Qualidade do Produto da ISO 25023 adaptado da ISO/IEC 25023 (2011)	21
Figura 3 – Arquitetura de um ambiente de <i>Data Warehousing</i>	33
Figura 4 – Exemplo de Esquema Estrela adaptado de Times (2012)	35
Figura 5 – Exemplo de Cubo de Dados	35
Figura 6 – Metodologia de Projeto de <i>Data Warehouse</i> proposta por Kimball e Ross (2002)	36
Figura 7 – Diagrama Dimensional de Árvore adaptado de Golfarelli, Maio e Rizzi (1998) e Sampaio (2007)	37
Figura 8 – Arquitetura do Ambiente de <i>Data Warehousing</i> para Métricas de Código- Fonte	42
Figura 9 – Diagrama de Árvore para Valor Percentil das Métricas de Código-Fonte	49
Figura 10 – Diagrama de Árvore para Avaliação de Cenários de Limpeza de Código- Fonte	49
Figura 11 – Diagrama de Árvore para avaliação da Taxa de Aproveitamento de Oportunidade de Melhoria de Código-Fonte	50
Figura 12 – Projeto Físico do <i>Data Warehouse</i>	51
Figura 13 – Interface do Kettle	52
Figura 14 – Interface Gráfica do Pentaho BI Platform	55
Figura 15 – Arquitetura Ambiente de <i>Data Warehousing</i> para Métricas de Código- Fonte	57
Figura 16 – Metodologia de Estudo de Caso proposta por Wohlin et al. (2012) . . .	58
Figura 17 – Informações do Repositório de Código-Fonte	63

Lista de tabelas

Tabela 1 – Objetivos Específicos do Trabalho	17
Tabela 2 – Objetivos Específico OE6	18
Tabela 3 – Modelo de Informação para métricas de código-fonte com base na ISO/IEC 15939 (2002)	20
Tabela 4 – Percentis para métrica NOM extraídos de Meirelles (2013)	25
Tabela 5 – Nome dos Intervalos de Frequência	26
Tabela 6 – Configurações para os Intervalos das Métricas para Java	27
Tabela 7 – Conceitos de Limpeza extraídos de Machini et al. (2010)	29
Tabela 8 – Cenários de Limpeza	31
Tabela 9 – Diferenças entre OLAP e OLTP extraído de Times (2012), Rocha (2000) e Neri (2002)	37
Tabela 10 – Exemplo do Total de Vendas de uma Rede de Lojas no mês de Novembro	39
Tabela 11 – Exemplo do Total de Vendas de uma rede de lojas no mês de novembro com a dimensão Produto	39
Tabela 12 – Exemplo do Total de vendas da Loja Norte no mês de novembro	39
Tabela 13 – Exemplo de Vendas por produto de uma rede de lojas nos meses de novembro e dezembro	40
Tabela 14 – Exemplo de Vendas do Produto A na rede de Lojas	40
Tabela 15 – Exemplo de Vendas por Loja para cada um dos Produtos nos meses de Novembro e Dezembro	40
Tabela 16 – Critérios Gerais de seleção de ferramentas	43
Tabela 17 – Critérios Específicos para Ferramenta de Análise Estática de Código- Fonte	43
Tabela 18 – Características do SonarQube e do Analizo	44
Tabela 19 – Análise do SonarQube e do Analizo quanto aos critérios gerais e quanto aos critérios específicos de ferramentas de análise estática	45
Tabela 20 – Requisitos de Negócio da Avaliação dos Cenários de Limpeza de Código- Fonte conforme as configurações especificadas na Tabela 6	46
Tabela 21 – Requisitos de Negócio da Avaliação de Cenários de Limpeza de Código- Fonte conforme a Tabela 8	47
Tabela 22 – Fatos e Dimensões do <i>Projeto de Data Warehouse</i>	48
Tabela 23 – Características do Kettle e avaliação quanto aos critérios gerais de se- leção de ferramentas	53
Tabela 24 – Características do Pentaho BI Platform e avaliação quanto aos critérios gerais de seleção de ferramentas	55

Tabela 25 – Características do Saiku Analytics e avaliação quanto aos critérios gerais de seleção de ferramentas	56
Tabela 26 – Objetivos Específicos de Estudo de Caso	59
Tabela 27 – Critérios de Seleção de Objeto do Estudo de Caso	59
Tabela 28 – Dados do Estudo de Caso	62

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>
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	15
1.1	Contexto	15
1.2	Problema	15
1.3	Questão de Pesquisa	15
1.4	Objetivos	16
1.5	Hipótese	17
1.6	Organização do Trabalho	18
2	MÉTRICAS DE SOFTWARE	19
2.1	Processo de Medição de Software	19
2.2	Classificação das Métricas de Software	20
2.3	Métricas de Código-Fonte	22
2.3.1	Métrica de Tamanho e Complexidade	22
2.3.2	Métricas de Orientação à Objetos	23
2.3.3	Configurações de Qualidade para Métricas de Código-Fonte	24
2.3.4	Código Limpo	28
3	DATA WAREHOUSING	32
3.1	<i>Extraction-Transformation-Load (ETL)</i>	33
3.2	<i>Data Warehouse</i>	34
3.2.1	Metodologia do Projeto do <i>Data Warehouse</i>	36
3.3	<i>On-Line Analytical Processing (OLAP)</i>	37
3.4	Visualização de Dados	40
4	AMBIENTE DE DATA WAREHOUSING PARA MÉTRICAS DE CÓDIGO-FONTE	42
4.1	Arquitetura do Ambiente <i>Data Warehousing</i> para Métricas de Código-Fonte	42
4.2	Ferramenta de Análise Estática de Código-Fonte	43
4.3	Projeto do <i>Data Warehouse</i>	45
4.4	Ferramentas de <i>Data Warehousing</i>	51
4.4.1	Implementação da Extração, Transformação e Carga dos Dados	52
4.4.2	Implementação das Consultas OLAP e Visualização de Dados	54
4.5	Resumo Ferramental do Ambiente de <i>Data Warehousing</i>	57
5	ESTUDO DE CASO	58

5.1	Planejamento do Estudo de Caso	58
5.1.1	Trabalhos Relacionados ao Estudo de Caso	58
5.1.2	Objetivos do Estudo de Caso	58
5.1.3	Seleção de Objeto do Estudo de Caso	59
5.1.4	Objeto de Estudo	60
5.1.4.1	Visão Geral	60
5.1.4.2	O Software Analisado	61
5.1.5	Dados, Fonte dos Dados e Forma de Análise dos Dados	61
5.1.6	Validade do Estudo de Caso	62
5.1.7	Limitação do Estudo de Caso	62
5.2	Execução do Estudo de Caso e Análise dos Dados	62
5.2.1	Análise dos Valores Percentis para Métricas de Código-Fonte do SICG	63
5.2.2	Análise dos Cenários de Limpeza identificados no SICG	63
5.2.3	Análise da Taxa de Aproveitamento de Oportunidade de Melhoria de Código-Fonte	63
6	CONCLUSÃO	64
6.1	Resultados da Implementação do Ambiente de <i>Data Warehousing</i>	64
6.2	Limitações e Trabalhos Futuros	64
	Referências	65
	APÊNDICE A – DESCRIÇÃO SIMPLICADA DO PROCESSO DE ETL NO KETTLE	70

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 trechos de código não coesos, que venham a se desfazer 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 está 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 pedaços são passíveis de eliminação com a refatoração, que é a técnica de modificação das estruturas internas do código-fonte sem modificação o 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, normamente, não apresentam associação entre os resultados numéricos 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 coletados e compartilhados 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 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 foi derivado em 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.	QP2 - Como automatizar o processo de medição de métricas de código-fonte?	M2 - Quantidade de 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 - 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?	<p>M5 - Taxa de Aproveitamento de Oportunidades de Melhoria de Código em uma release é calculada como:</p> $T_r = \frac{\sum_{i=1}^n Ce_i}{\sum_{i=1}^n Cl_i}$ <p>onde Ce é o cenário de limpeza e Cl é o Número de classe</p>

Tabela 1 – Objetivos Específicos do Trabalho

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), mostraram 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 hipotetizado a hipótese H1:

H1: A construção de um ambiente de *Data Warehousing* traz automa-

ção necessário ao processo de medição de métricas de código-fonte, possibilidade de criar um campo de conhecimento semântico para facilitar a interpretação de métricas de código-fonte, trazer visibilidade necessária às métricas de código-fonte e inferir 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 visualizações de métricas de código-fonte providas pelo ambiente de <i>Data Warehousing</i> em um projeto.	QE6 - Quantas formas diferentes de visualização das métricas de código-fonte são possíveis em um ambiente de <i>Data Warehousing</i> ?	M6 - Quantidade de formas diferentes de visualizar cada métrica de código-fonte

Tabela 2 – Objetivos Específico OE6

1.6 Organização do Trabalho

Após a leitura da introdução, encontrar-se-á adiante o Capítulo 2 que apresenta 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 a projeto 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 foi acompanhado as 24 releases do software SICG (Sistema Integrado de Conhecimento e Gestão) do Instituto de Patrimônio Histórico e Artístico Nacional (IPHAN). 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, a fim de obter produtos de informação para cada necessidade de informação. Para isto, cada necessidade de informação, que é uma situação que requer conhecimento com intuito de gerenciar 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 fim de 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 ou medidas, 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 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

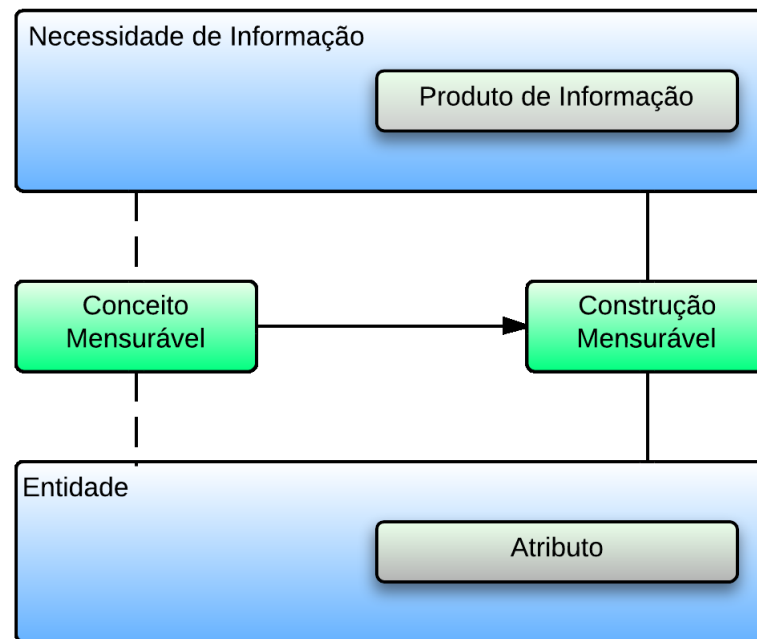


Figura 1 – Modelo de Informação da ISO 15939

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 às necessidades de informação. Quando se trata da coleta 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 a entidade é mapeada ([ISO/IEC 15939, 2002](#)). As escalas podem ser:

- **Nominal:** A medição é categórica. Nesta escala, só é possível a 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, podendo estas serem *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:

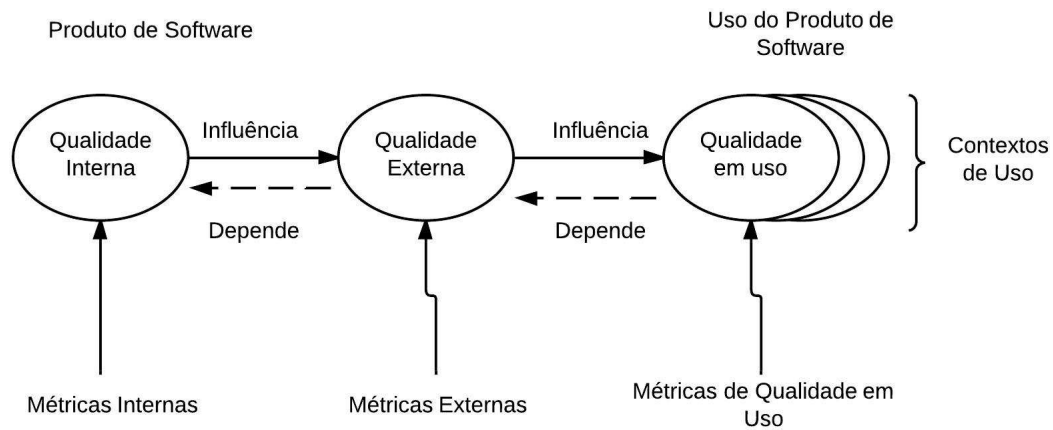


Figura 2 – Modelo de Qualidade do Produto da ISO 25023 adaptado da [ISO/IEC 25023 \(2011\)](#)

- **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. Exemplos de atributos da qualidade externa: correção, usabilidade, eficiência e robustez. Qualidade externa mede o comportamento do software. Estas 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 implantação.

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 significa que as métricas de código-fonte são métricas objetivas e possuem 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) Essa medida 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

A evolução dos paradigmas de programação permitiu que as linguagens de programação assumissem diversas características entre si. O paradigma da orientação à objetos permitiu aos programadores abstrair computação ao negócio, isso significou evolução no desenvolvimento quando comparado ao paradigma procedural e permanência na academia quanto na indústria de desenvolvimento de software ([LI; HENRY, 1993](#)).

Dado a grande utilização do paradigma no desenvolvimento de software, foram selecionadas algumas das principais métricas de orientação à 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 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 \(1995\)](#), que propôs a LCOM4. 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. Essa métrica é usada para ajudar a identificar o potencial de reúso 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, boas práticas de programação recomendam que os atributos de uma classe devem ser manipulados através dos métodos de acesso ([BECK, 1997](#))
- **RFC** (*Response For a Class* - Respostas para uma Classe) é 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, 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, 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, atingindo o 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 considerar dois cenários, considerou-se dois software como referência 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	(Open JDK8)	(Tomcat)
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" que 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 de coorelação métricas constitui um cenário de limpeza, onde é mostrado a correlação de um "conceito de limpeza", apresentados 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 **OpenJDK** 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 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	Objetivos durante a Refatoração	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 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 reponsáveis 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, não intrusiva ao trabalho da equipe de desenvolvimento e que estas possam mostrar trazer a visibilidade da qualidade do código-fonte produzido pela equipe de desenvolvimento durante um determinado período de tempo (dias, meses, anos).

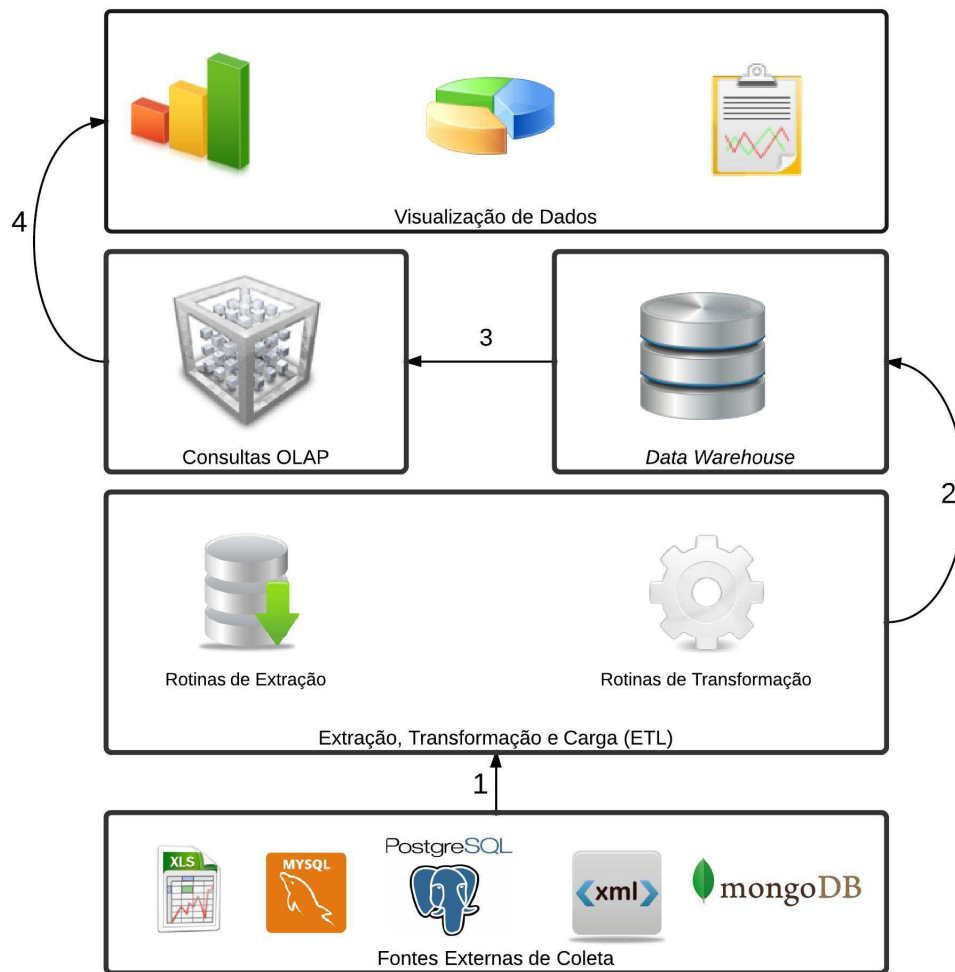


Figura 3 – Arquitetura de um ambiente de *Data Warehousing*

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.

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 dis-

tintas, 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 com uma área de armazenamento intermediária entre fontes e o *data warehouse*. Normalmente, 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 no 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 expressam 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. Quando em um modelo dimensional, se faz necessário uso da normalização, o modelo passa então a ser chamado por *modelo 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.

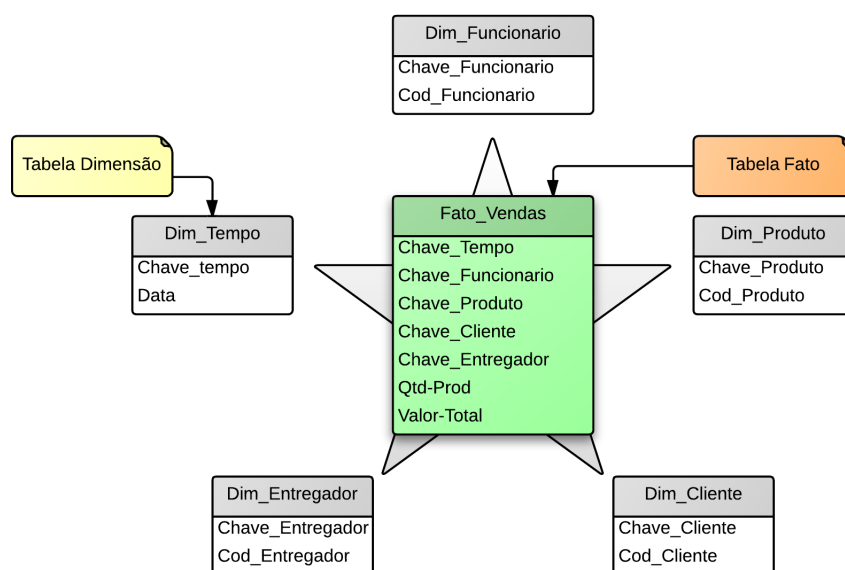


Figura 4 – Exemplo de Esquema Estrela adaptado de Times (2012)

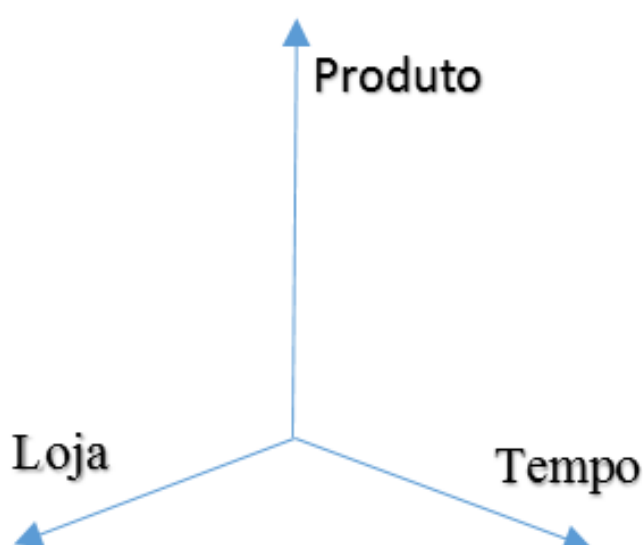


Figura 5 – Exemplo de Cubo de Dados

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 (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. Contudo, 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 ainda considerar 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 da metodologia proposta por Kimball e Ross (2002), para o projeto de 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 tais como cálculos específicos que podem vir a ser entradas para o processo de Extracion-Transformation-Load. No segundo passo, a identificação da periodicidade de coleta dos dados é essencial para coleta correta e agregação dos dados em níveis ou hierarquias. O terceiro e quarto passos resultam por fim no modelo dimensional que foi apresentado na seção 3.2.

Tendo os fatos e dimensões identificados pela de Kimball e Ross (2002), é possível construir um diagrama dimensional de árvore tal como exemplo da Figura 7. Este diagrama foi proposto por Golfarelli, Maio e Rizzi (1998) com intuito de conceber conceituais de **Data Warehouse**, onde 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.

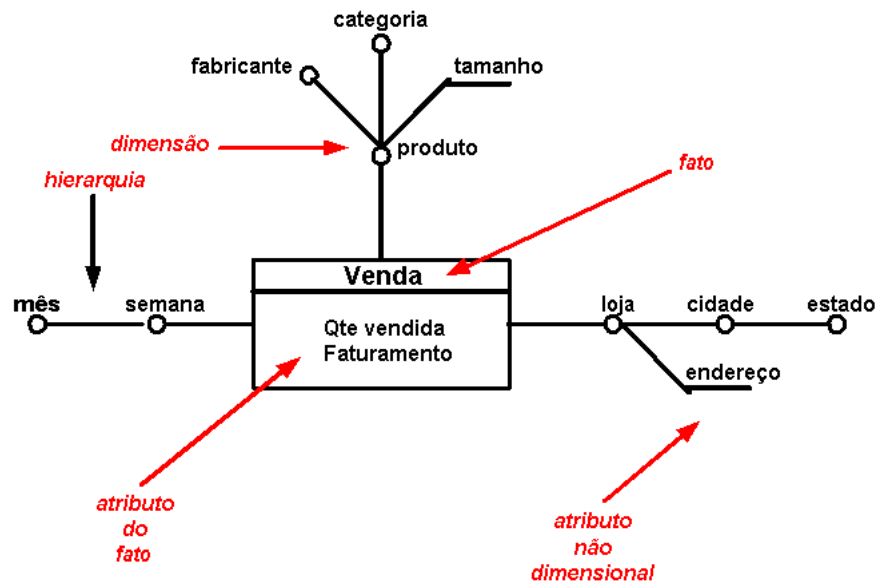


Figura 7 – Diagrama Dimensional de Árvore adaptado de [Golfarelli, Maio e Rizzi \(1998\)](#) e [Sampaio \(2007\)](#)

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 normalizados, visando a eliminação da redundância de dados.

As principais diferenças das operações *On-Line Analytical Processing* (OLAP) para as operações *Online Transaction Processing* (OTLP) são apresentados 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 extraído 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 $P, F_1(m_1), \dots, F_p(m_p)$
From $C(D_1(A_{11}), \dots, D_n(A_{n+1}))$
Where $\phi(D_1)$ and ... and $\phi(D_n)$
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 trazer à evidência para os dados (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 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 sua similaridades e dissimilaridades.

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 se mostra Figura 8.

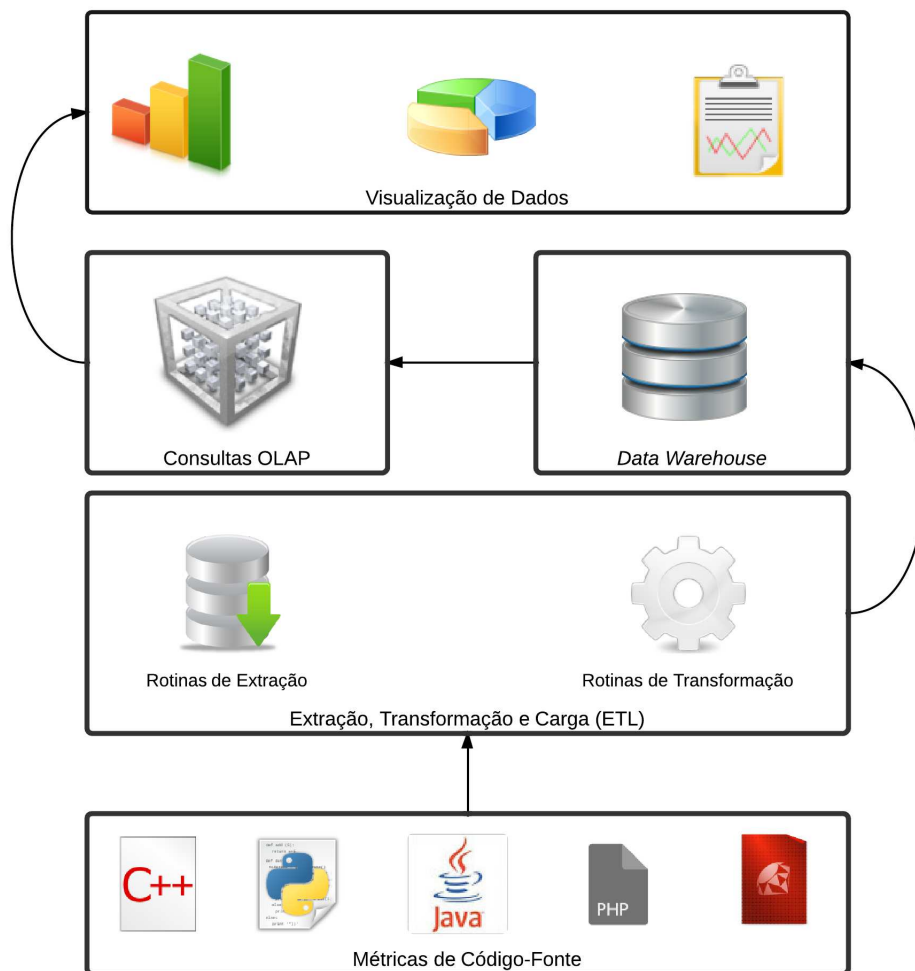


Figura 8 – Arquitetura do Ambiente de *Data Warehousing* para Métricas de Código-Fonte

Para selecionar as ferramentas, que implementarão 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 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 pre-selecionados o SonarQube ¹ e Analizo ² cujas principais características de ambas são apresentadas na Tabela 18.

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

² Disponível em <<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

Tendo as características gerais de cada ferramenta levantadas, 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 fase inicial do trabalho, fora feita a análise entre o Analizo e o SonarQube, que resultou na decisão de utilizar o SonarQube. Contudo, desde a versão 4.1, o SonarQube retirou as métricas: **LCOM4**, **RFC**, **DIT**, **NOC** ⁵. Dado que este fato, impacta sobre o principal objetivo do trabalho, migrou-se 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, foram identificados os processos de negócio e seus respectivos requisitos como se mostra nas Tabelas 20 e 21

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

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 valor percentil obtido para cada métrica de código-fonte em uma determinada release do projeto para a configuração Open JDK8 Metrics
RN4	Comparar o valor percentil obtido 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 o valor percentil obtido 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 Cenários de Limpeza 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	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 conforme a Tabela 8

Aplicando o segundo passo da metodologia de [Kimball e Ross \(2002\)](#), identificou-se a periodicidade 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 uma 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 à aditividade ⁶ e relacionados com as respectivas dimensões conforme se mostra na Tabela 22

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

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

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 9, 10 e 11.

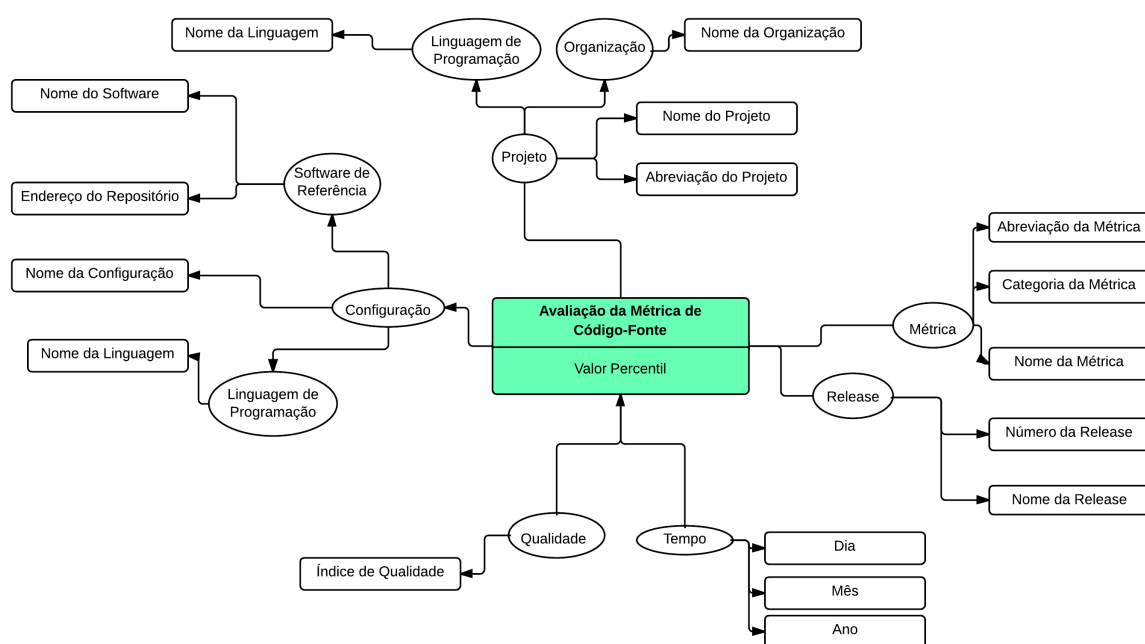


Figura 9 – Diagrama de Árvore para Valor Percentil das Métricas de Código-Fonte

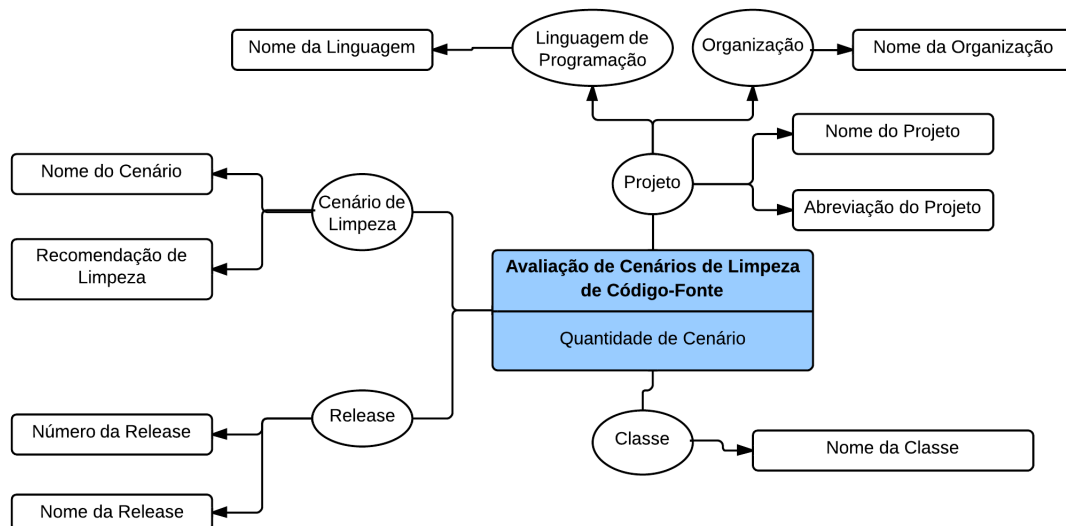


Figura 10 – Diagrama de Árvore para Avaliação de Cenários de Limpeza de Código-Fonte

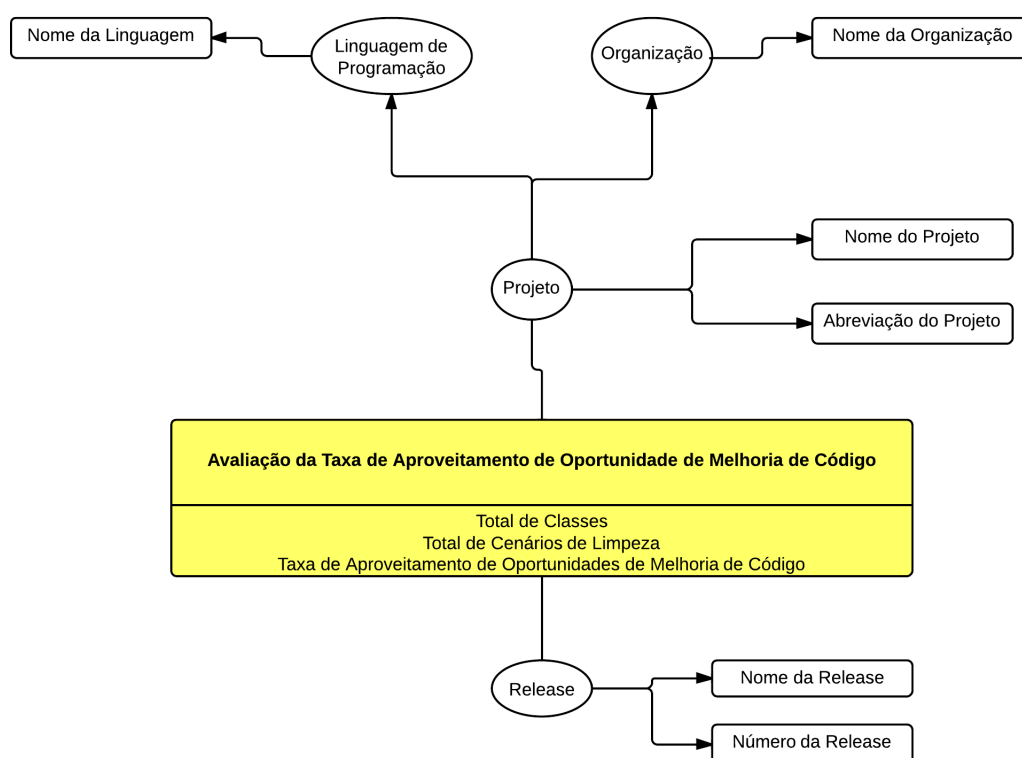
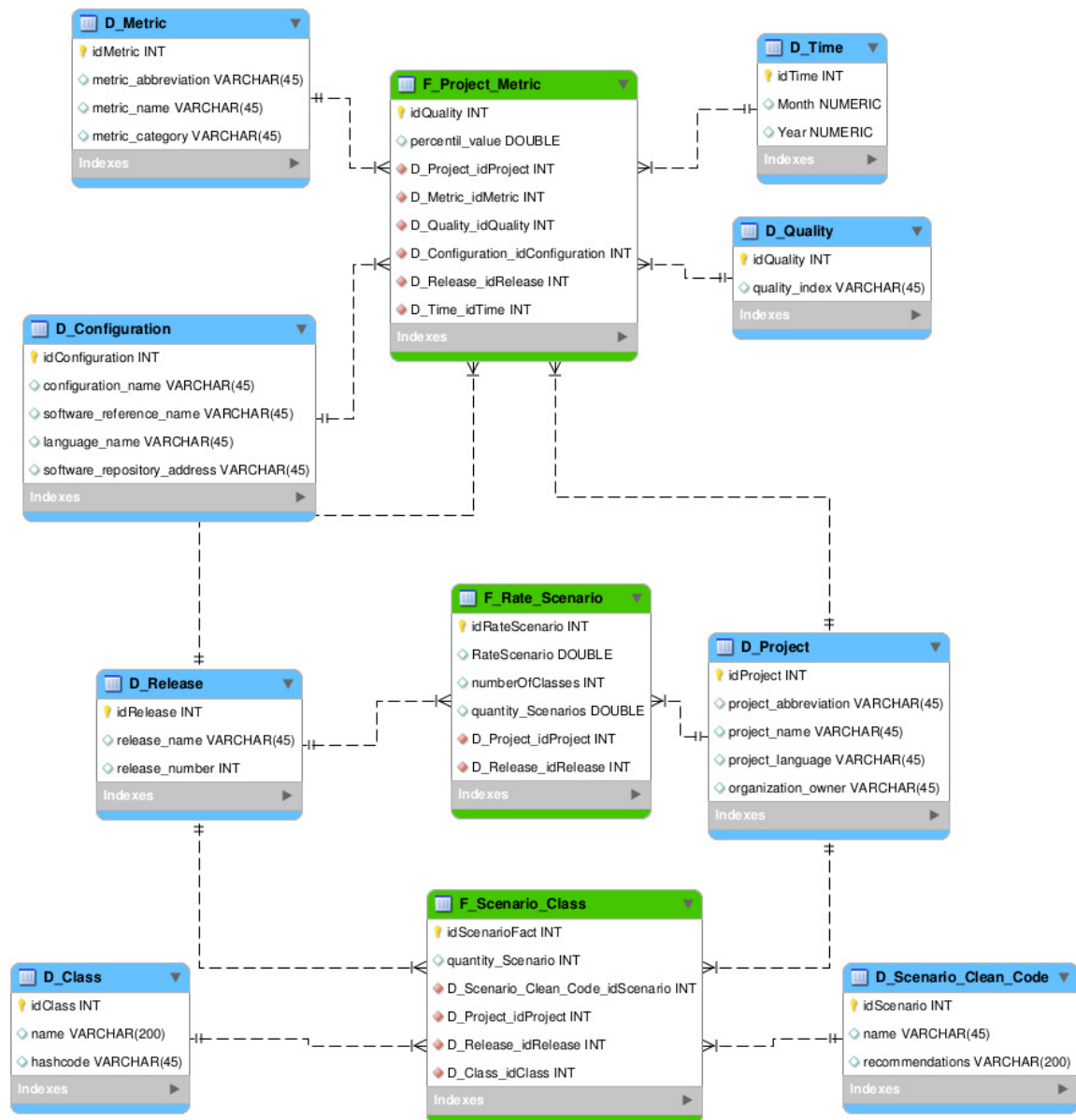


Figura 11 – Diagrama de Árvore para avaliação da Taxa de Aproveitamento de Oportunidade de Melhoria de Código-Fonte

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 12, cada digrama de árvore foi traduzido para uma tabela fato (tabelas verdes) com as respectivas dimensões (tabelas azuis)

Figura 12 – Projeto Físico do *Data Warehouse*

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 BI Suite Community Edition. Este apresenta soluções que cobrem as áreas de ETL, *reporting*, OLAP e mineração de dados. Cada um dos componentes utilizados 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). A interface do Kettle é mostrada na Figura 13 e as principais características do Kettle e a análise quanto aos critérios gerais de seleção de ferramentas são apresentadas na Tabela 23.

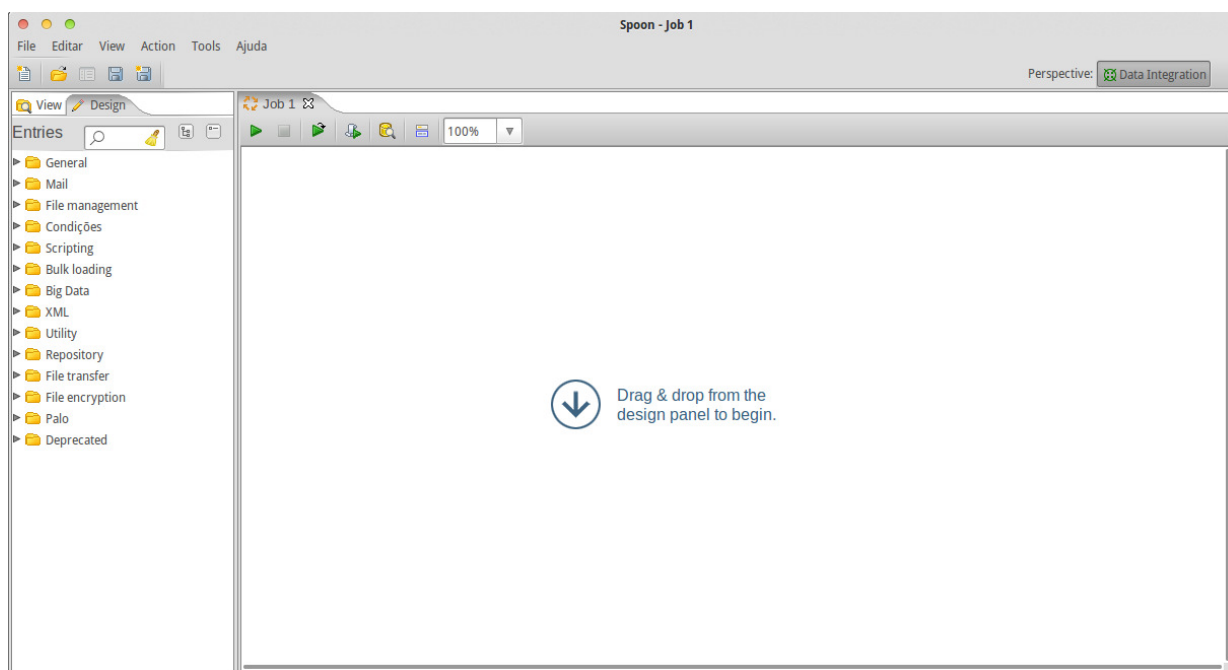


Figura 13 – Interface do Kettle

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


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				
Última 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

O Kettle possui dois tipos de componentes internos: *Job* e *Transformation*. O primeiro permite 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. Já a *Transformation* permite tratamento aos dados incluindo desde entrada de dados por diversas fontes até a persistência em uma variedade de SGBDs.

Para a implementação do ETL no Kettle, utilizou-se 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 1.


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

Código-Fonte 1 – Parser de CSV para JSON

Todas as implementações de *Transformation* e *Job* que foram elaboradas no decorrer do trabalho foram descritas em detalhes no apêndice A.

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, cuja interface inicial é apresentada na Figura 14, tem as principais características e a análise quanto aos critérios gerais de seleção de ferramentas são apresentadas na Tabela 24.

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

Característica		CG01	CG02	CG03	CG04
					
Licença	Apache License 2.0	✓			
Última 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

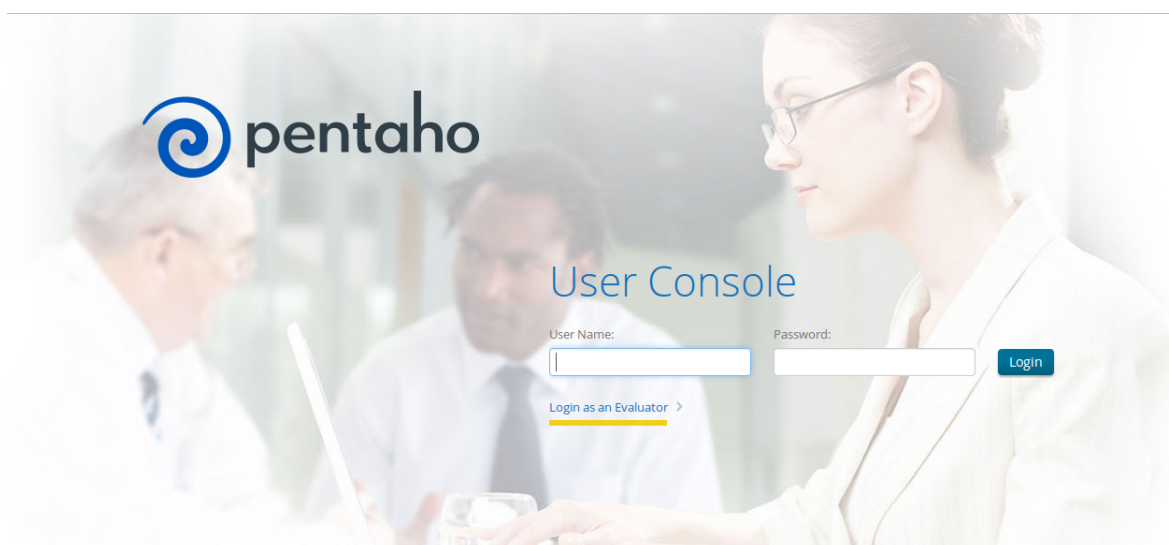


Figura 14 – Interface Gráfica do Pentaho BI Platform

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 e 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				
Última 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 seguem o modelo dimensional, tal como mostra o exemplo do trecho de Código-Fonte 2.

```

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

```

Código-Fonte 2 – Exemplo de *Query* em linguagem MDX

4.5 Resumo Ferramental do Ambiente de *Data Warehousing*

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

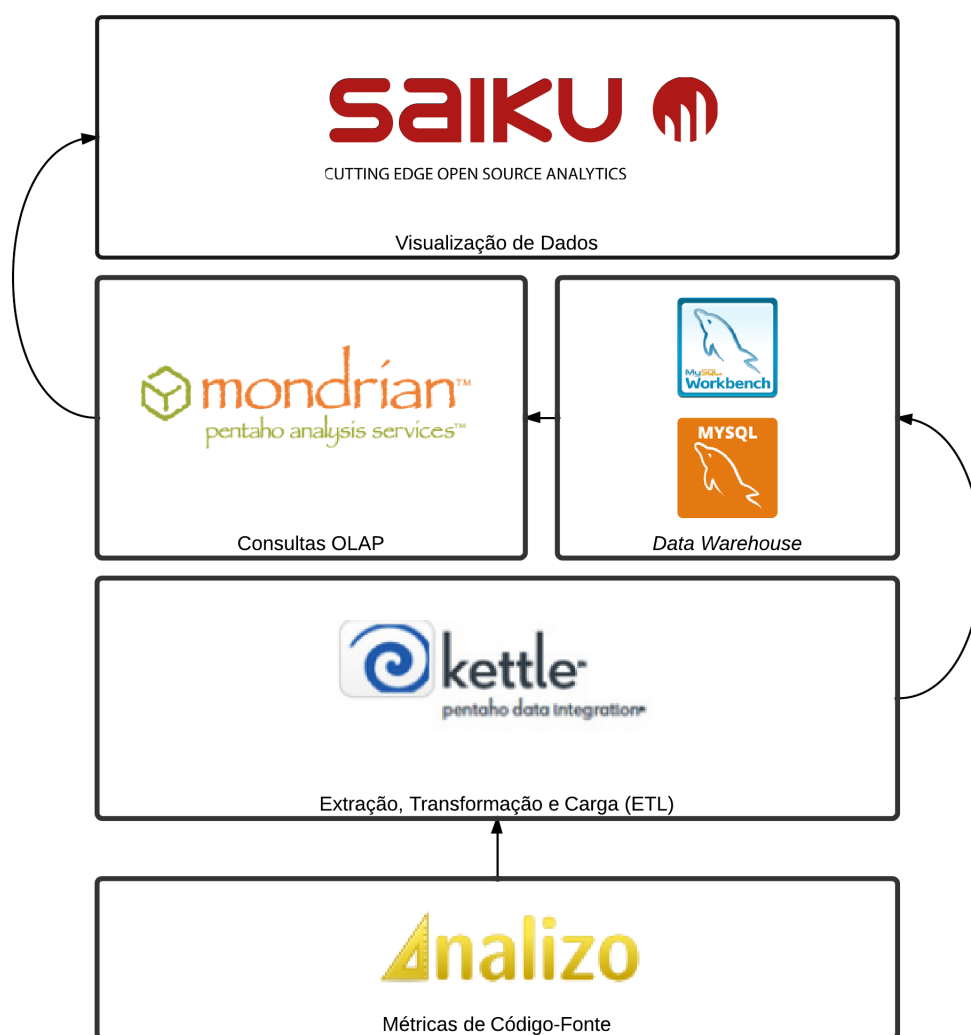


Figura 15 – Arquitetura Ambiente de *Data Warehousing* para Métricas de Código-Fonte

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 Wheezy

5 Estudo de Caso

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

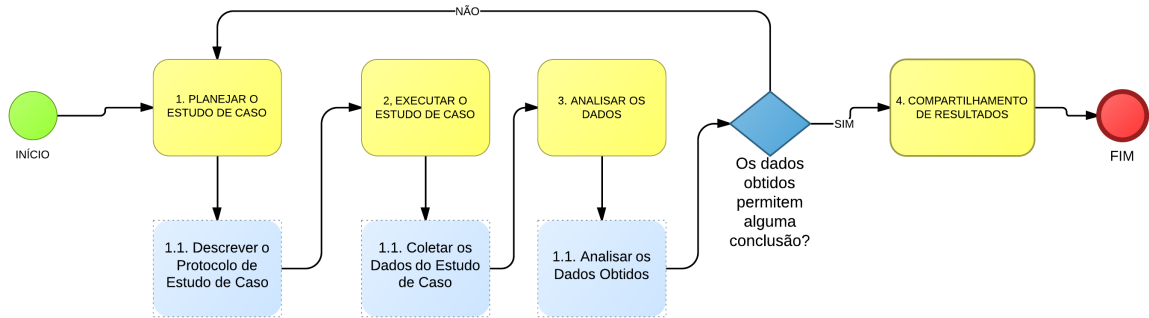


Figura 16 – Metodologia de Estudo de Caso proposta por Wohlin et al. (2012)

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), descrever 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 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*. 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	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 de métricas de código-fonte observadas no ambiente de <i>Data Warehousing</i>	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 do objeto do estudo de caso considerou-se 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 afim de observar o comportamento de dados ao longo do tempo.
CEC3	Disponibilidade do código-fonte para que o Analizo 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

5.1.4.1 Visão Geral

O IPHAN é 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 cadastramento, 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 cadastramento 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 cadastramento 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 frustra os esforços para 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 decorrentes do intervalo entre ação e recepção das respostas, ou seja, entre a percepção do problema e sua solução.

Apesar do fato de os processos de cadastramento, normatização de sítios urbanos tombados e fiscalização de bens imóveis de metodologia 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, cadastramento, 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

Os principais dados quantitativos e qualitativos foram identificados na Tabela 28.

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.
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 Cálculo	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 apresentasse valores mais altos com passar o tempo, significa dizer que o projeto não está aproveitando as oportunidades de refatoração
Formas de Visualização das métricas de código-fonte no Ambiente de <i>Data Warehousing</i>	Número de formas diferentes de visualização das métricas para cada requisito de negócio	Para cada requisito de negócio, conta-se as diferentes formas possíveis de visualização das métricas de código-fonte.

Tabela 28 – Dados do Estudo de Caso

5.1.6 Validade do Estudo de Caso

5.1.7 Limitação do Estudo de Caso

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.

6 Conclusão

6.1 Resultados da Implementação do Ambiente de *Data Warehousing*

Discutir o nível de visibilidade das métricas de código-fonte, os benefícios da abordagem de DWing para Métricas

6.2 Limitações e Trabalhos Futuros

Limitações: Ambiente dependente da ferramenta de análise estática, Abordagens para as métricas, Utilização do Mezuro ou BiggData

Sugestões do Professor Paulo - Trabalho Futuro: Melhor Verificar a correlação entre os padrões de projeto e os cenários de limpeza. Verifica a adoção das práticas de refatoração e revisão de código conjuntamente com a detecção de cenários

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 16 e 19.
- BASILI, V. R.; ROMBACH, H. D. *TAME: Integrating Measurement into Software Environments*. 1987. Disponível em: <<http://drum.lib.umd.edu/handle/1903/7517>>. Citado na página 23.
- BECK, K. *Smalltalk Best Practice Patterns. Volume 1: Coding*. [S.l.]: Prentice Hall, Englewood Cliffs, NJ, 1997. Citado na página 24.
- BECK, K. Embracing change with extreme programming. *Computer*, v. 32, n. 10, p. 70–77, 1999. ISSN 0018-9162. Citado na página 47.
- BECK, K. *Extreme Programming Explained*. [S.l.]: Addison Wesley, 1999. Citado na página 15.
- BECK, K. *Test-driven development: by example*. [S.l.]: Addison-Wesley Professional, 2003. Citado na página 15.
- BECK, K. *Implementation patterns*. [S.l.]: Pearson Education, 2007. Citado 3 vezes nas páginas 15, 28 e 30.
- 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 17 e 58.
- 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 58.
- 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 17 e 58.
- CHAUDHURI, S.; DAYAL, U. An overview of data warehousing and olap technology. *ACM Sigmod record*, ACM, v. 26, n. 1, p. 65–74, 1997. Citado 2 vezes nas páginas 32 e 38.
- 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 23.
- 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 16.

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 19.

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 37.

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 15.

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 19, 20 e 21.

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 17 e 58.

FONSECA, R.; SIMOES, A. Alternativas ao xml: Yaml e json. 2007. Citado na página 53.

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

GAMMA, E. et al. *Design patterns: elements of reusable object-oriented software*. [S.l.]: Pearson Education, 1994. Citado na página 30.

GARDNER, S. R. Building the. *Communications of the ACM*, v. 41, n. 9, p. 53, 1998. Citado na página 32.

GOLFARELLI, M.; MAIO, D.; RIZZI, S. Conceptual Design of Data Warehouses from E/R Schemes. 1998. Citado 4 vezes nas páginas 9, 36, 37 e 48.

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 32.

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 22.

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

INMON, W. H. *Building the Data Warehouse*. New York, NY, USA: John Wiley & Sons, Inc., 1992. ISBN 0471569607. Citado 3 vezes nas páginas 34, 36 e 38.

ISO/IEC 15939. *ISO/IEC 15939: Software Engineering - Software Measurement Process*. [S.l.], 2002. Citado 4 vezes nas páginas 10, 19, 20 e 21.

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

JONES, T. C. *Applied Software Measurement: Assuring Productivity and Quality*. New York: McGraw-Hill, 1991. Citado na página 22.

KIMBALL, R.; ROSS, M. *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling*. 2nd. ed. New York, NY, USA: John Wiley & Sons, Inc., 2002. ISBN 0471200247, 9780471200246. Citado 9 vezes nas páginas 9, 33, 34, 35, 36, 37, 38, 45 e 47.

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 22.

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 23.

LORENZ, M.; KIDD, J. *Object-Oriented Software Metrics*. [S.l.]: Prentice Hall, 1994. Citado na página 24.

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 10, 28, 29 e 30.

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 15 e 30.

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 15 e 30.

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 28 e 30.

MCCABE, T. J. A Complexity Measure. *IEEE Transactions Software Engineering*, v. 2, n. 4, p. 308–320, December 1976. Citado na página 22.

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 23.

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 9 vezes nas páginas 10, 15, 20, 21, 23, 24, 25, 26 e 44.

- 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 21 e 22.
- 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 40.
- 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 15 e 30.
- 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 15 e 30.
- 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 15 e 30.
- 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 10 e 37.
- 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 15.
- 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 17 e 58.
- 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 15 e 30.
- 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 10, 32, 37, 38 e 39.
- 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 24.
- 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 17 e 58.

- 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 37.
- 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 24.
- 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 23.
- 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 17, 32 e 58.
- SOMMERVILLE, I. *Software Engineering*. 9. ed. Harlow, England: Addison-Wesley, 2010. ISBN 978-0-13-703515-1. Citado na página 15.
- SPOFFORD, G. et al. *MDX Solutions with Microsoft SQL Server Analysis Services 2005 and Hyperion Essbase*. [S.l.]: Wiley Pub., 2006. Citado na página 56.
- 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, 10, 34, 35, 37 e 39.
- WICHMANN, B. et al. Industrial perspective on static analysis. *Software Engineering Journal*, 1995. Citado na página 15.
- WOHLIN, C. et al. *Experimentation in software engineering*. [S.l.]: Springer, 2012. Citado 2 vezes nas páginas 9 e 58.
- 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 15.

APÊNDICE A – Descrição Simplificada do Processo de ETL no Kettle

Adicionar os arquivos de transformação e Job