



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

Extração e Visualização de Métricas de Código-Fonte em um ambiente de *Data Warehousing*

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

Brasília, DF
2013



Guilherme Baufaker Rêgo

Extração e Visualização de Métricas de Código-Fonte em um ambiente de *Data Warehousing*

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

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: Prof. Msc. Hilmer Rodrigues Neri

Coorientador: Prof. Dr. Paulo Roberto Miranda Meirelles

Brasília, DF

2013

Guilherme Baufaker Rêgo

Extração e Visualização de Métricas de Código-Fonte em um ambiente de *Data Warehousing*/ Guilherme Baufaker Rêgo. – Brasília, DF, 2013-
57 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 , 2013.

1. Métricas de Código-Fonte. 2. DWing. I. Prof. Msc. Hilmer Rodrigues Neri. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Extração e Visualização de Métricas de Código-Fonte em um ambiente de *Data Warehousing*

CDU 02:141:005.6

Guilherme Baufaker Rêgo

Extração e Visualização de Métricas de Código-Fonte em um ambiente de *Data Warehousing*

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

Trabalho aprovado. Brasília, DF, 05 de dezembro de 2013:

Prof. Msc. Hilmer Rodrigues Neri
Orientador

Prof. Dr. Paulo Roberto Miranda Meirelles
Coorientador

Prof. Dr. Rodrigo Bonifácio de Almeida
Convidado 1

Titulação e Nome do Professor
Convidado 02
Convidado 2

Brasília, DF
2013

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 aos 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 a oportunidade concedida no projeto Desafio Positivo, que foi um divisor de águas em minha formação profissional quanto a oportunidade de conhecer desde *data warehouse*, metodologias ágeis e outras áreas importantes da engenharia de software.

Agradeço também ao Prof. Paulo Meirelles por ter compartilhado tanto o conhecimento sobre métricas, ferramentas, testes de software quanto por ter compartilhado suas ponderações muito importantes para o meu crescimento pessoal e profissional.

Sem Prof. Hilmer e o Prof. Paulo este trabalho não seria possível.

Agradeço 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 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 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 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 aos amigos Vinícius Vieira Meneses, Renan Costa Filgueiras, Luiz Mattos e Marcos Ramos do LAPPIS - Laboratório Avançado de Produção, Pesquisa e Inovação em Software - por todo conhecimento repassado a mim e pelas horas de reflexão sobre questões cruciais do desenvolvimento de software.

*‘The only way of discovering the limits of the possible is to venture a little way past them
into the impossible.’
(Arthur C. Clarke)*

Resumo

O resumo deve ressaltar o objetivo, o método, os resultados e as conclusões do documento. A ordem e a extensão destes itens dependem do tipo de resumo (informativo ou indicativo) e do tratamento que cada item recebe no documento original. O resumo deve ser precedido da referência do documento, com exceção do resumo inserido no próprio documento. (...) As palavras-chave devem figurar logo abaixo do resumo, antecedidas da expressão Palavras-chave:, separadas entre si por ponto e finalizadas também por ponto. O texto pode conter no mínimo 150 e no máximo 500 palavras, é aconselhável que sejam utilizadas 200 palavras. E não se separa o texto do resumo em parágrafos.

Palavras-chaves: latex. abntex. editoração de texto.

Abstract

This is the english abstract.

Key-words: latex. abntex. text editoration.

Lista de ilustrações

Figura 1 – Modelo de Informação da ISO 15939 extraído de Gomes (2011)	17
Figura 2 – Modelo de Qualidade do Produto da ISO 25023 adaptado de ISO/IEC 25023 (2011)	19
Figura 3 – Arquitetura de um ambiente de <i>Data Warehousing</i> extraído de Rocha (2000)	24
Figura 4 – Exemplo de Esquema Estrela extraído de Times (2012)	26
Figura 5 – Exemplo de Cubo Multidimensional de dados extraído de Times (2012)	27
Figura 6 – Exemplo de Dashboard extraído de Minardi (2013)	32
Figura 7 – Outro exemplo de Dashboard extraído de Minardi (2013)	32
Figura 8 – Arquitetura do Ambiente de Dwing para Métricas de Código-Fonte . .	33
Figura 9 – Comparação entre JSON, YAML e XML extraído de Fonseca e Simoes (2007)	37
Figura 10 – Modelo multidimensional do <i>Data Warehouse</i>	39
Figura 11 – Interface do Kettle	40

Lista de tabelas

Tabela 1 – Modelo de Informação para metodologias ágeis com base na ISO/IEC 15939 (2002)	18
Tabela 2 – Nome dos Intervalos de Frequência	22
Tabela 3 – Intervalos das Métricas para Java e C++	23
Tabela 4 – Diferenças entre OLAP e OLTP extraído de Times (2012), Rocha (2000) e Neri (2002)	28
Tabela 5 – Exemplo do Total de Vendas de uma Rede de Lojas no mês de Novembro	29
Tabela 6 – Exemplo do Total de Vendas de uma rede de lojas no mês de novembro com a dimensão Produto	29
Tabela 7 – Exemplo do Total de vendas da loja norte no mês de novembro	29
Tabela 8 – Exemplo de Vendas por produto de uma rede de lojas nos meses de novembro e dezembro	30
Tabela 9 – Exemplo de Vendas do Produto A na rede de lojas	30
Tabela 10 – Exemplo de Vendas por loja para cada um dos produtos nos meses de novembro e dezembro	31
Tabela 11 – Critérios Gerais de escolha das ferramentas	34
Tabela 12 – Critérios Específicos para Ferramenta de Análise Estática de Código-Fonte	34
Tabela 13 – Características do SonarQube e do Analizo	35
Tabela 14 – Análise do SonarQube e do Analizo quanto aos critérios gerais e quanto aos critérios específicos de ferramentas de análise estática	36
Tabela 15 – Características do Kettle e avaliação quanto aos critérios gerais de seleção de ferramentas	41

Lista de abreviaturas e siglas

ACC	<i>Afferent Connections per Class</i>
ACCM	<i>Average Cyclomatic Complexity per Method</i>
DIT	<i>Depth of Inheritance Tree</i>
DW	<i>Data Warehouse</i>
DWing	<i>Data Warehousing</i>
ETL	<i>Extraction-Transformation-Load</i>
GQM	<i>Goal-Question-Metric</i>
IEC	<i>International Electrotechnical Commission</i>
ISO	<i>International Organization for Standardization</i>
LCOM4	<i>Lack of Cohesion in Methods</i>
LOC	<i>Lines of Code</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>
XP	<i>eXtreme Programming</i>

Sumário

1	Introdução	14
1.1	Objetivos	15
1.1.1	Objetivo Geral	15
1.1.2	Objetivos Específicos	15
1.2	Metodologia de Pesquisa	16
1.3	Organização do Trabalho	16
2	Métricas de Software	17
2.1	Processo de Medição de Software	17
2.2	Classificação das Métricas de Software	18
2.3	Métricas de Código-Fonte	19
2.3.1	Métrica de Tamanho e Complexidade	20
2.3.2	Métricas de Orientação à Objetos	20
2.3.3	Intervalos da Métricas	21
3	Data Warehousing (DWing)	24
3.1	<i>Extraction-Transformation-Load</i> (ETL)	25
3.2	<i>Data Warehouse</i>	25
3.2.1	Metodologia do Projeto do <i>Data Warehouse</i>	27
3.3	<i>On-Line Analytical Processing</i> (OLAP)	27
3.4	Visualização de Dados	31
4	Implementação do Ambiente de DWing	33
4.1	Arquitetura da Implementação	33
4.2	Ferramenta de Análise Estática de Código-Fonte	34
4.2.1	Acompanhamento das Métricas de Código-Fonte de um Software Livre	37
4.3	Projeto do <i>Data Warehouse</i>	38
4.4	Ferramentas de Dwing	39
4.4.1	Implementação da Extração, Transformação e Carga dos Dados	40
4.4.2	Implementação das Consultas OLAP e Visualização de Dados	41
5	Considerações Finais	42
5.1	Sobre o Trabalho	42
5.1.1	Resultados Parciais	42
5.1.2	Trabalhos Futuros	42

Referências 43

APÊNDICE A Implementações no Kettle 47

APÊNDICE B SQL do *Data Warehouse* 48

1 Introdução

Dentre as inúmeras características que fazem um bom software, várias delas podem ser percebidas no código-fonte e algumas são exclusivas dele (MEIRELLES, 2013). Logo decisões de desenvolvedores, que impactam fortemente sobre a qualidade do código-fonte, podem gerar trechos de código não coesos, que venham a se desfazer com o tempo, que aumentam exponencialmente a chance de manutenções corretivas onerosas (MEIRELLES, 2013) (BECK, 2007) (BECK, 1999b).

Portanto avaliar continuamente a qualidade do código-fonte, pode identificar antecipadamente trechos a serem melhorados, atingindo assim maiores graus de manutenibilidade, coesão, entendimento e legibilidade (FOWLER, 1999). Entre os vários métodos de análise da qualidade, está a análise estática de código-fonte, que se refere a uma análise automatizada das estruturas internas do código, em nível de linguagem de programação, sem realizar a execução do mesmo, com objetivo de obter métricas de código-fonte (TERRA; BIGONHA, 2008) (EMANUELSSON; NILSSON, 2008) (WICHMANN et al., 1995) (NIELSON; NIELSON; HANKIN, 1999) (SOMMERVILLE, 2010).

Durante anos, vários trabalhos foram publicados visando definição formal de métodos de análise estática de código, como por exemplo, os trabalhos de Wichmann et al. (1995), Nielson, Nielson e Hankin (1999) e Emanuelsson e Nilsson (2008). Contudo as ferramentas que realizam este procedimento sobre o código-fonte apresentam alguns problemas, tais como:

- P1 - Ausência de resultados consolidados do produto, pois a maior parte das métricas é extraída de elementos internos menores (Bibliotecas, Pacotes, Classes, Métodos) do código-fonte.
- P2 - Ausência de mecanismos de tratamento, separação, recuperação, organização e persistência de dados.
- P3 - Ausência de associação entre resultados numéricos e forma de interpretá-los: Ferramentas de análise estática frequentemente mostram seus resultados como valores numéricos isolados para cada métrica (MEIRELLES, 2013).
- P4 - Em grande parte das ferramentas, a visualização dos resultados não é agradável, isto é, são apresentados um conjunto de dados em uma janela terminal contendo os valores das métricas.

Os problemas enunciados anteriormente trazem muitos prejuízos às organizações que utilizam processos de aferição de qualidade de código-fonte como um indicador do

desenvolvimento de bons produtos de software, pois sem possibilidade de manter registros das métricas de código-fonte, torna-se inviável qualquer análise temporal da evolução da qualidade do produto de software.

Dado este contexto, é crucial que dados relacionados às métricas sejam coletados e compartilhados entre projetos e pessoas em uma visão organizacional unificada, para que determinada organização ou time 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).

Vários trabalhos têm mostrado que ambientes de *Data Warehousing* (DWing) são boas soluções para atender à visão organizacional unificada de métricas de software proposta por Chulani et al. (2003). Vide 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). Tendo os trabalhos, enunciados anteriormente, como norteadores, adotou-se como hipótese de pesquisa deste trabalho que a **visualização e a extração de métricas de código-fonte em ambientes de DWing possibilitam as equipes maior poder de análise sobre a qualidade do código fonte quando comparada com as análises providas por ferramentas de análise estática de código-fonte isoladamente.**

1.1 Objetivos

Esta seção apresenta o objetivo geral e os objetivos específicos deste Trabalho de Conclusão de Curso.

1.1.1 Objetivo Geral

Sob o prisma da hipótese de pesquisa, há como objetivo geral a proposição e construção de um ambiente de *Data Warehousing*, para extrair e visualizar métricas de código-fonte com objetivo de que a interpretação das métricas de código-fonte venha a ser facilitada.

1.1.2 Objetivos Específicos

Os objetivos específicos deste trabalho são:

OE1 - Construir o ambiente de Dwing para extração e visualização de métricas de código-fonte utilizando ferramentas de software livre.

- OE2 - Analisar a qualidade de um software livre afim de validar a utilização do ambiente de DWing para métricas de código-fonte.
- OE3 - Incorporar indicadores qualitativos para cada métrica de código-fonte.
- OE4 - Facilitar o entendimento das métricas de código-fonte, por meio de apresentação, em formas visuais, das análises obtidas por meio das métricas de código-fonte

1.2 Metodologia de Pesquisa

A metodologia de pesquisa deste trabalho foi constituída de pesquisa documental para a construção do referencial teórico e experimental para construção do ambiente de DWing instanciado em um caso de avaliação de qualidade de código-fonte de software livre escolhido.

1.3 Organização do Trabalho

Para a primeira fase deste Trabalho de Conclusão de Curso, além desta introdução, o texto foi organizado em capítulos. O Capítulo 2 apresenta as métricas de software bem como o processo de medição, descrito pela ISO 15939, e as métricas de código-fonte. O Capítulo 3 apresenta conceitualmente o ambiente de *data warehousing* (DWing). O Capítulo 4 apresenta a implementação do ambiente de DWing construída neste trabalho. Por fim, o capítulo 5 apresenta as considerações finais com uso do ambiente de DWing na extração e visualização de métricas de código-fonte.

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, afim 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, custo afim de mapear um atributo, que é uma 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, sendo que este é utilizado por abordagens mais antigas como GQM em [Basili, Caldiera e Rombach \(1996\)](#), que é uma outra abordagem para medição, contudo compreende-se que o termo medida tem valor semântico equivalente ao de métrica no contexto do deste trabalho

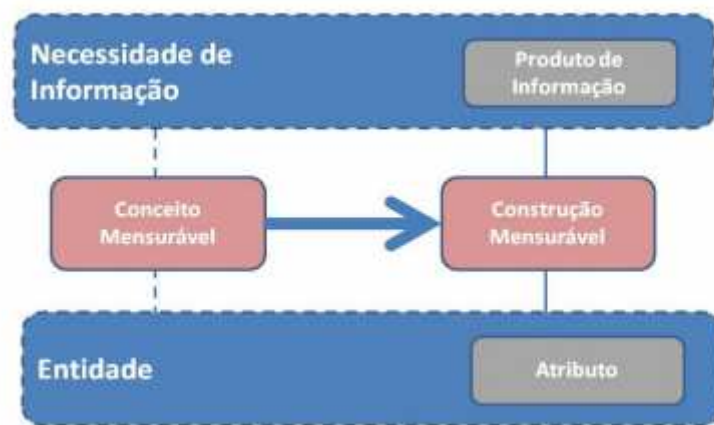


Figura 1 – Modelo de Informação da ISO 15939 extraído de [Gomes \(2011\)](#)

O modelo de informação da [ISO/IEC 15939 \(2002\)](#) é utilizado para construir o propósito geral da medições e a identificação de medidas ou métricas que respondem as necessidades de informação. Em metodologias ágeis, como por exemplo, é possível construir um modelo de informação tal como na Tabela 1.

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 1 – Modelo de Informação para metodologias ágeis 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 as quais entidade é mapeada ([ISO/IEC 15939, 2002](#)). As escalas podem ser:

- Nominal: A medição é categórica. Nesta escala, só é possível realização de comparações, sendo que a ordem não possui significado ([ISO/IEC 15939, 2002](#)) ([FENTON; PFLEEGER, 1998](#)) ([MEIRELLES, 2013](#)).
- Ordinal: A medição é baseada em ordenação, ou seja os valores possuem ordem, mas a distância entre eles não possui significado. Por exemplo, nível de experiência dos programadores ([ISO/IEC 15939, 2002](#)) ([FENTON; PFLEEGER, 1998](#)) ([MEIRELLES, 2013](#)).

- Intervalo: A medição é baseada em distâncias iguais definidas para as menores unidade. Por exemplo, o aumentar 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. Como 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 a 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:

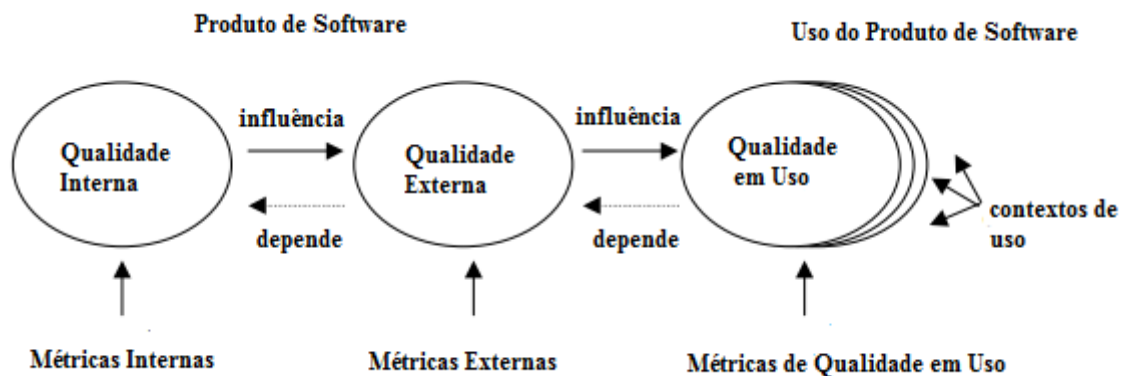


Figura 2 – Modelo de Qualidade do Produto da ISO 25023 adaptado de ISO/IEC 25023 (2011)

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).

Portanto as métricas de código-fonte que são obtidas por meio da análise estática de código-fonte são obtidas por meio de especificações executáveis do software, isto é, trechos de código escritos em alguma linguagem de programação, são métricas objetivas e tem características como validade, simplicidade, objetividade, fácil obtenção e robustez (MILLS, 1999). Meirelles (2013) realizou um levantamento sistemático na literatura das métricas de código-fonte. Estas foram categorizadas, nas seguintes categorias, apresentadas nas subseções a seguir.

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 é impresso. A segunda lei de Lehman (1980) enuncia que a complexidade aumenta à medida que o software é evoluído, a menos que seja um trabalho de manutenção. 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 as principais 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 Ciclômática 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 ciclômática de um de um método.

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 funcional, por exemplo, enxerga o programa como uma sequência de funções que podem ser executadas em modo funcional. Já o paradigma orientado a objetos visa abstrair as unidades computacionais em Classes, que representam em grande parte do desenvolvimento, unidades reais do negócio, partindo destas abstrai-se instâncias computacionais, isto são, objetos propriamente ditos. A seguir são apresentadas as principais métricas de orientação à objetos:

ACC (*Afferent Connections per Class*) Conexões Aferentes por Classe é 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).

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).

LCOM4 (*Lack of Cohesion in Methods*) Falta de Coesão entre Métodos. Originalmente proposto por Chidamber e Kemerer (1994) como LCOM, contudo essa não teve uma grande aceitabilidade. Após críticas e sugestões a métrica 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.

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).

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).

NOC (*Number of Children*) Número de Filhos é o número 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.

2.3.3 Intervalos da Métricas

Em sua tese Meirelles (2013) analisou as métricas do código-fonte de 38 projetos de software livre, em um total de 344.872 classes das aplicações mais bem sucedidas (*Chrome, Firefox, OpenJDK, VLC e entre outros*) e percebeu que há certos valores que são frequentemente encontrados quando se analisa o código-fonte de aplicações que utilizam a mesma linguagem de programação. Meirelles (2013), após uma análise estatística com o 75º percentil dos valores obtidos para cada métrica de código-fonte, classificou estes intervalos, em *muito frequente, frequente, pouco frequente e não frequente*.

Visando o melhor entendimento das métricas, resolveu-se renomear tal como a Tabela 2. Posteriormente, é apresentada a Tabela 3, decorrente do estudo de Meirelles (2013), com os intervalos encontrados para C++ e Java.

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

Tabela 2 – Nome dos Intervalos de Frequência

Métrica	Indicador	Java	C++
LOC	Excelente	[de 0 a 33]	[de 0 a 31]
	Bom	[de 34 a 87]	[de 32 a 84]
	Regular	[de 88 a 200]	[de 85 a 207]
	Ruim	[acima de 200]	[acima de 207]
ACCM	Excelente	[de 0 a 2,8]	[de 0 a 2,0]
	Bom	[de 2,9 a 4,4]	[de 2,1 a 4,0]
	Regular	[de 4,5 a 6,0]	[de 4,1 a 6,0]
	Ruim	[acima de 6]	[acima de 6]
ACC	Excelente	[de 0 a 1]	[de 0 a 2,0]
	Bom	[de 1,1 a 5]	[de 2,1 a 7,0]
	Regular	[de 5,1 a 12]	[de 7,1 a 15]
	Ruim	[acima de 12]	[acima de 15]
RFC	Excelente	[de 0 a 9]	[de 0 a 29]
	Bom	[de 10 a 26]	[de 30 a 64]
	Regular	[de 27 a 59]	[de 65 a 102]
	Ruim	[acima de 59]	[acima de 102]
LCOM4	Excelente	[de 0 a 3]	[de 0 a 5]
	Bom	[de 4 a 7]	[de 6 a 10]
	Regular	[de 8 a 12]	[de 11 a 14]
	Ruim	[acima de 12]	[acima de 14]
NOM	Excelente	[de 0 a 8]	[de 0 a 10]
	Bom	[de 9 a 17]	[de 11 a 17]
	Regular	[de 18 a 27]	[de 18 a 26]
	Ruim	[acima de 27]	[acima de 26]
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]
NOC	Excelente	[de 0 a 1]	[0]
	Bom	[de 1 a 2]	[1]
	Regular	[de 2 a 3]	[de 1 a 2]
	Ruim	[acima de 3]	[acima de 2]

Tabela 3 – Intervalos das Métricas para Java e C++

Os intervalos, que foram apresentados na Tabela 3 serão utilizados como indicadores de qualidade de código-fonte, de modo a simplificar a interpretação da qualidade do código-fonte no componente de visualização de dados, do ambiente de *Data Warehousing*, a ser apresentado a seguir.

3 *Data Warehousing* (DWing)

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

Data Warehousing (DWing) é uma coleção de tecnologias de suporte à decisão disposta a capacitar os responsáveis por tomar decisões a fazê-las de forma mais rápida (CHAUDHURI; DAYAL, 1997 apud ROCHA, 2000). Em outras palavras, trata-se de um processo para montar e gerenciar dados vindos de várias fontes, com o objetivo de prover uma visão analítica de parte ou todo o 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 a qualidade total do código-fonte produzido pela equipe durante um determinado período de tempo (dias, meses, anos).

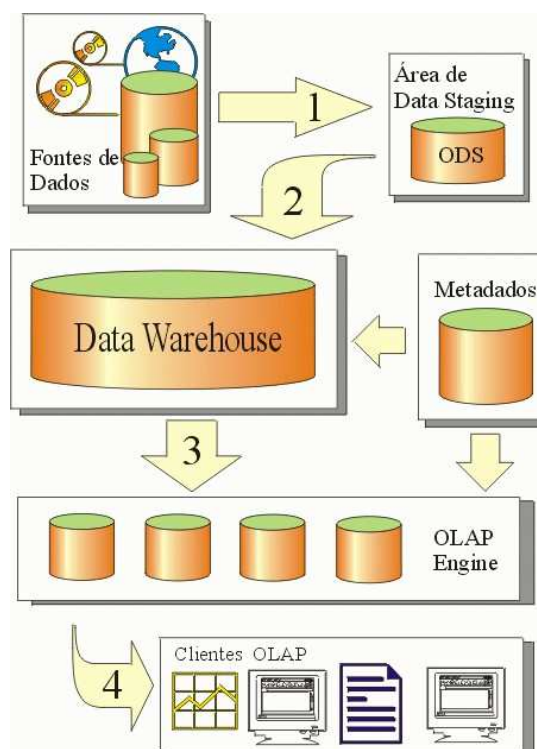


Figura 3 – Arquitetura de um ambiente de *Data Warehousing* extraído de Rocha (2000)

A Figura 3 descreve uma arquitetura geral de um ambiente de DWing, de tal forma que,

- i) As setas 1 e 2 representam o processo de *Extraction-Transformation-Load*;
- ii) A seta 3 representa as consultas *On-Line Analytical Processing (OLAP)*;
- iii) 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 podendo consumir até 85% de todo o esforço em um DWing (KIMBALL; ROSS, 2002).

- Extração: No ambiente de *data warehousing*, os dados, que provêm de fontes distintas tais como planilhas, bases relacionais em diferentes tipos de banco de dados (MySQL, Oracle, Postgres 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 afim de serem utilizados pelas consultas OLAP.

3.2 *Data Warehouse*

Data Warehouse (DW) 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 DWing (KIMBALL; ROSS, 2002).

A necessidade de centralização e agregação dos dados em um *data warehouse* mos-

trou 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 desnormalizadas, 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

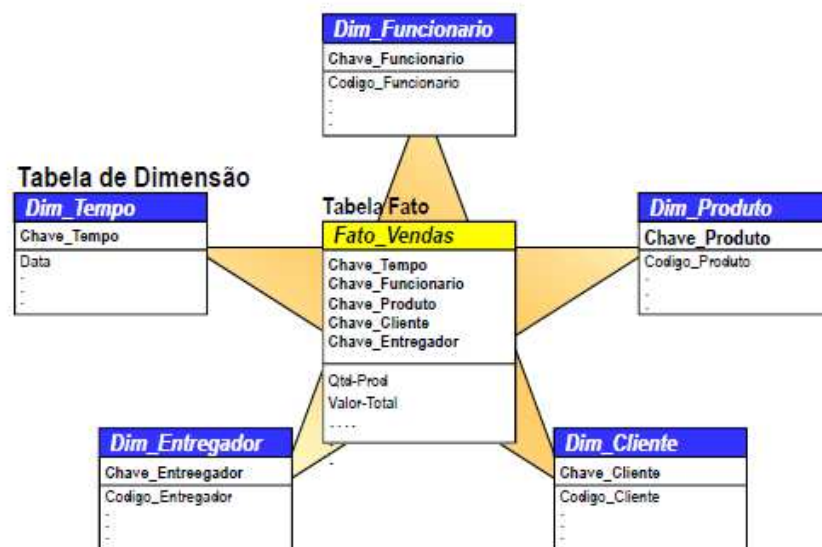


Figura 4 – Exemplo de Esquema Estrela extraído de Times (2012)



Figura 5 – Exemplo de Cubo Multidimensional de dados extraído de [Times \(2012\)](#)

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*

[Kimball e Ross \(2002\)](#) enuncia que o ambiente de DWing nasce na necessidade do negócio e logo o projeto de um *data warehouse* deve seguir os seguintes passos:

- 1) Selecionar o Processo de Negócio com requisito fundamental do projeto DW;
- 2) Verificar a periodicidade de coleta dos dados do processo de negócio (diários, semanais, mensais, trimestrais, semestrais ou anuais);
- 3) Identificar as dimensões;
- 4) Identificar os fatos;

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 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 4.

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 4 – 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 à 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 lojas, que agregam as Lojas Sul, Norte e Oeste tal como se mostra a Tabela 5, a operação Drill-Down pode ser exemplificada, quando se adiciona a dimensão Produto na Tabela 5, isto é, aumentando o nível de detalhes, tendo então como resultado a Tabela 6. Já a operação de Roll-Up é o contrário, isto é diminuir o nível de detalhe partindo da Tabela 6 para Tabela 5.

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

Tabela 5 – 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 6 – 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 5, obtendo assim a Tabela 7.

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

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

Tabela 7 – Exemplo do Total de vendas da loja norte no mês de novembro

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 8, obtendo assim a Tabela 9

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 8 – 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 9 – 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 afim de restringir a visualização das dimensões em uma tabela. (ROCHA, 2000). A operação de Pivoting pode ser exemplificada ao partir da Tabela 8 para Tabela 10

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 10 – 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, sendo que i) tabelas são utilizadas para consulta de valores individuais que podem ser comparados envolvendo, em certos casos, mais de uma unidade de medida; ii) gráficos são indicados para exibição de informação quantitativa na qual os valores indicam pontos de interesse e estes podem ser comparados por sua similaridades e dissimilaridades.

Few (2009) destaca que ainda que técnicas tradicionais de visualização de dados tais como, tabelas, histogramas, gráficos de pizza, podem descrever quase todos os tipos da relação quantitativa tais como, séries temporais, desvio, parte-todo, distribuição, etc; as representações tradicionais não são suficientes para transmitir ao receptor efetivamente a mensagem que dados transmitem. Além disso, Few (2009) enuncia que as técnicas tradicionais de visualização não tem nenhum poder de flexibilização quanto a visualização de dados.

Uma das técnicas de visualização de dados que tem se destacado é o uso de *dashboards*, que é uma representação visual das informações mais importantes necessárias para atingir um ou mais objetivos que cabe inteiramente em uma tela de computador de forma que podem ser monitoradas simultaneamente (MINARDI, 2013), tal como se mostra na Figura 6 e na Figura 7.

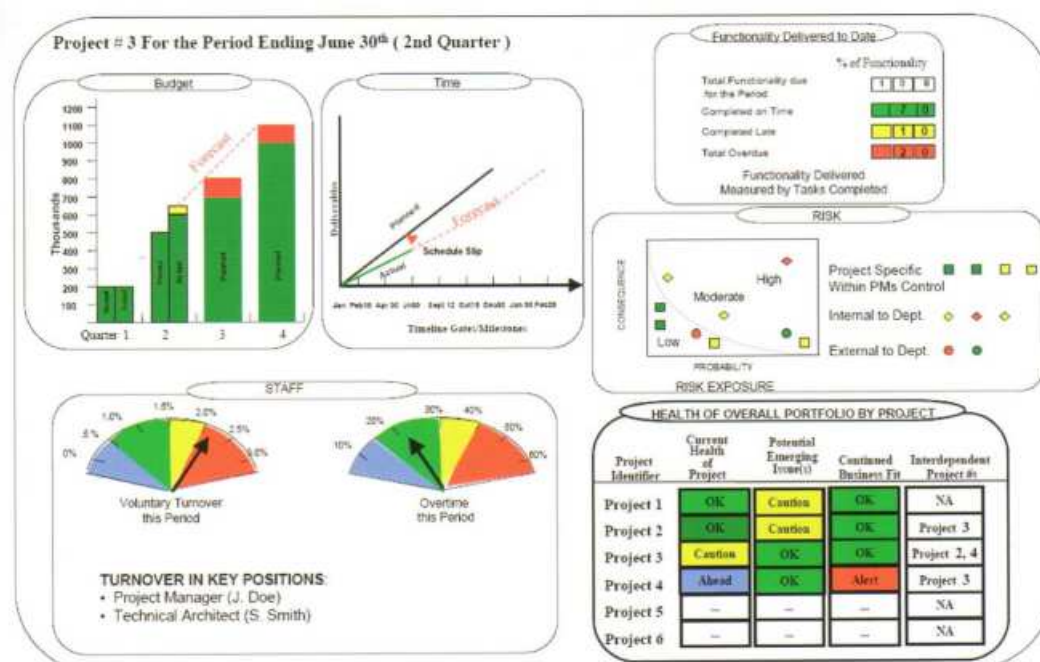


Figura 6 – Exemplo de Dashboard extraído de Minardi (2013)

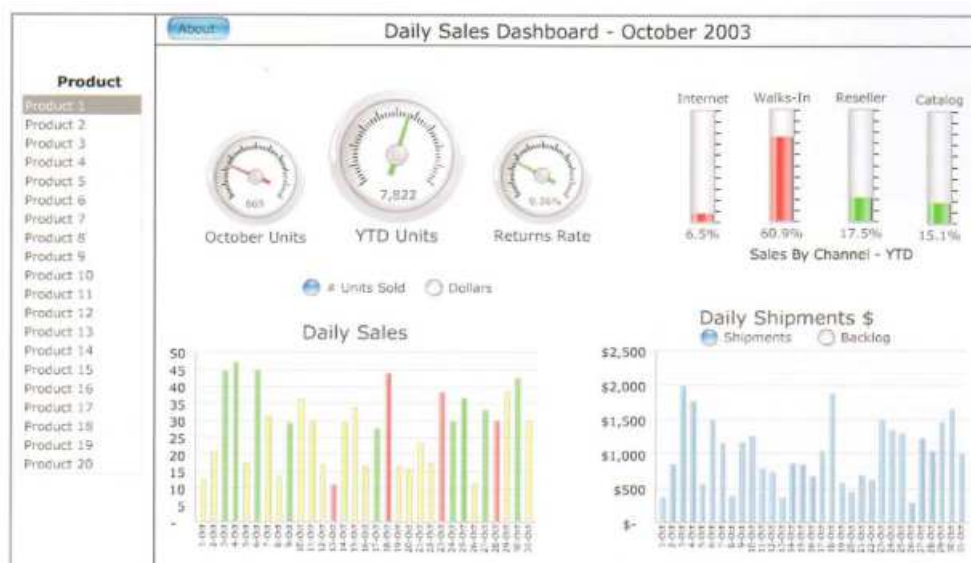


Figura 7 – Outro exemplo de Dashboard extraído de Minardi (2013)

Em um próximo passo do trabalho, pretende-se implementar o *Dashboard* para a visualização das métricas de código-fonte, dado que segundo Minardi (2013) um *dashboard* tem como papel estratégico a apresentação dos principais indicadores para tomada de decisão, logo comunicando os dados de maneira mais eficiente quando comparado com a exibição tabelas e gráficos isolados.

4 Implementação do Ambiente de DWing

4.1 Arquitetura da Implementação

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

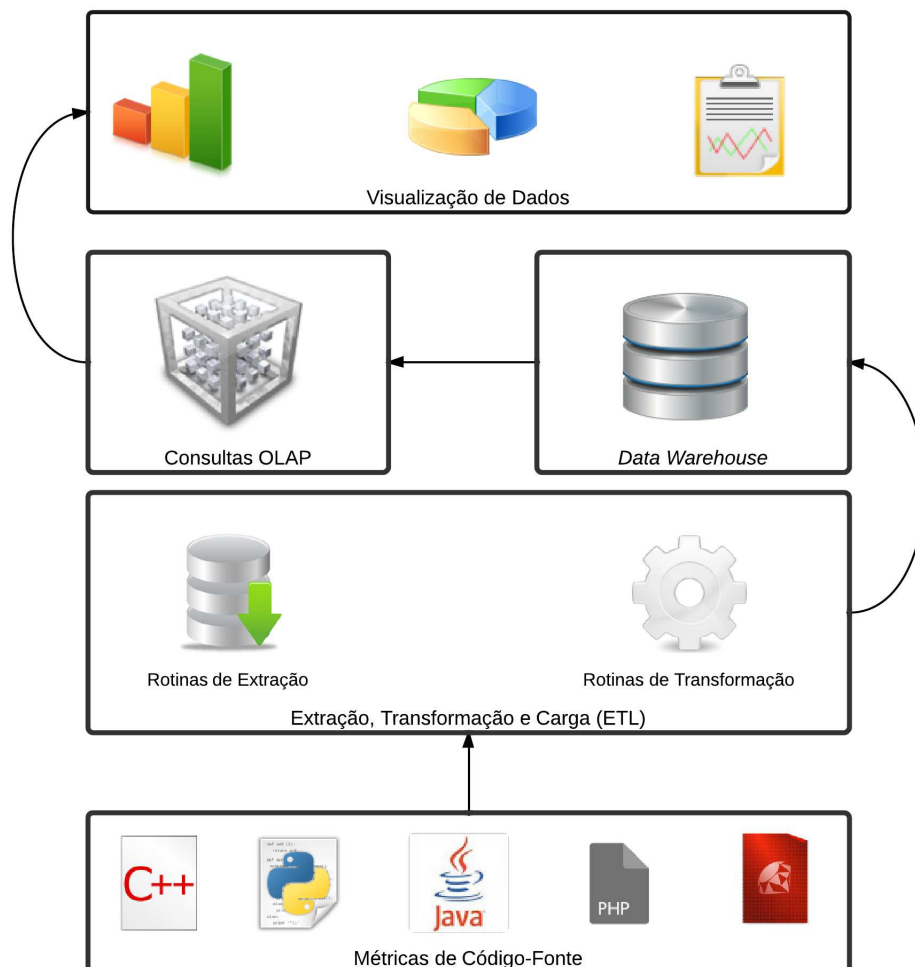


Figura 8 – Arquitetura do Ambiente de Dwing para Métricas de Código-Fonte

Para selecionar as ferramentas, que implementarão cada um dos componentes do ambiente DWing, estabeleceram-se critérios gerais de seleção tal como pode ser visto na Tabela 11.

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 11 – Critérios Gerais de escolha das 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 são as fontes externas de coleta, estabeleceram-se os critérios específicos para seleção de ferramentas de análise estática de código fonte (CAE) apresentados na Tabela 12

Identificador	Critério
CAE01	A ferramenta deve prover as métricas de código-fonte para as linguagens de programação, tal como especificado na Tabela 3.
CAE02	A ferramenta deve possuir saída de dados em arquivo em alguns dos seguintes formatos: JSON, XML, TXT, CSV.
CAE03	A ferramenta deve ser multiplataforma.

Tabela 12 – 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 13.

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

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



Característica		
Linguagens com Suporte	C, C++, Java	C, C++, Java, PHP, Scala, Python, Delphi, Pascal, Flex, ActionScript, Javascript, Groovy ³
Licença	GNU GPL3	GNU LGPL3
Formato de Saída das Métricas	YAML	JSON, XML
Plataforma	Windows, Linux, Mac OS X e Servidores de Aplicação Java	Debian e Ubuntu
Integração com outras ferramentas	Mezuro, Kalibro	Jenkins, Hudson, Mantis, JIRA, Crowd e entre outros
Números do Repositório Oficial no GitHub em 14/11/2013	<i>Commits</i> : 639	<i>Commits</i> : 7532
	<i>Branches</i> : 1	<i>Branches</i> : 16
	Contribuidores: 7	Contribuidores: 18
	<i>Releases</i> : 27	<i>Releases</i> : 36
Número de Casos Abertos no <i>Issue Tracker</i> em 14/11/2013	26	552
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 14/11/2013	1.17.0	4.0
Data de Lançamento da Última Versão Estável	31 de Janeiro de 2013	4 de Novembro de 2013

Tabela 13 – 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 14.

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

Critérios	SonarQube	Analizo
CG01	✓	✓
CG02	✓	✓
CG03	✓	✓
CG04	✓	✓
CAE01	✓	✓
CAE02	✓	✓
CAE03	✓	✗

Tabela 14 – Análise do SonarQube e do Analizo quanto aos critérios gerais e quanto aos critérios específicos de ferramentas de análise estática

Dado a comparação, percebe-se que ambas as ferramentas cumprem o propósito geral de levantar as métricas de código-fonte, com a diferença que SonarQube possui um suporte a mais linguagens de programação quando comparado com o Analizo, contudo dado que o escopo da análise de [Meirelles \(2013\)](#) se restringe a Java e C++, logo a falta de suporte à outras linguagens que o SonarQube suporta não é fator excludente da utilização do Analizo.

Percebe-se, ainda devido aos critérios específicos de seleção de ferramentas de análise estática, que o Analizo tem uma plataforma específica, sendo que o SonarQube é multiplataforma. Contudo, entende-se que como há objetivo de construir um ambiente de *Data Warehousing* utilizando ferramentas livres, logo poder-se-ia escolher também uma plataforma livre tal como o Ubuntu ou Debian.

A escolha foi ponderada então pelo formato de saída do arquivo. [Fonseca e Simoes \(2007\)](#) realizou um estudo experimental entre XML, JSON (Formatos de Saída do SonarQube) e YAML (Formato de Saída do Analizo), que é mostrado na Figura 9.

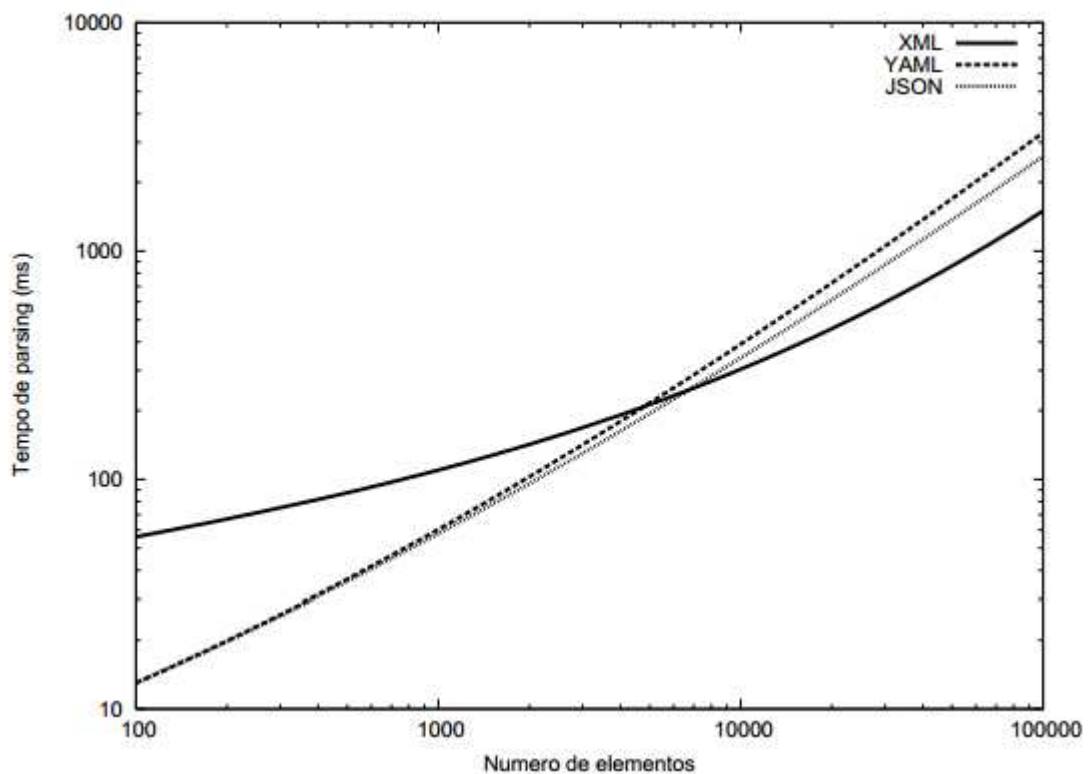


Figura 9 – Comparação entre JSON, YAML e XML extraído de [Fonseca e Simoes \(2007\)](#)

Após a análise da Figura 9, percebe-se que o YAML perde em desempenho para JSON e XML quando o número de elementos aumenta gradativamente. Em projetos de software, é natural que estes cresçam cada vez mais, tal como especifica a segunda lei de [Lehman \(1980\)](#), logo o formato JSON ou XML é mais adequado, fazendo com que a ferramenta SonarQube seja mais indicada para o contexto deste trabalho.

Entre JSON e XML, [Fonseca e Simoes \(2007\)](#) conclui que o transporte de dados é mais rápido utilizando JSON, pois se permite transportar um objeto serializado arbitrariamente complexo, e transformá-lo num objeto JavaScript sem grandes problemas, enquanto que o XML se deve construir um *parser* para interpretar os dados. Por este motivo, foi escolhido para extração das métricas de código-fonte neste trabalho, o formato JSON provido pela ferramenta SonarQube.

4.2.1 Acompanhamento das Métricas de Código-Fonte de um Software Livre

Com o objetivo de validar a implementação do ambiente de DWing para métricas de código-fonte, escolheu-se nesta etapa do trabalho de conclusão de curso, acompanhar a qualidade de um software livre conforme especificado pelo objetivo específico OE2.

Dado que os intervalos constituídos da análise de [Meirelles \(2013\)](#), apresentados na seção 2.3.3, são para as linguagens de programação C++ e Java, há então a restritividade

quanto ao acompanhamento de projetos feitos nessas linguagens.

Após uma leitura da documentação do SonarQube, descobriu-se que este mantém um repositório de análises de código-fonte de projetos de software livre, chamado Nemo ⁴. O Nemo disponibiliza projetos feitos nas linguagens Java, Javascript, Groovy e entre outros. Limitando-se apenas a linguagem Java, dado que não foi encontrado nenhum projeto C++ no repositório com grande frequência de atualizações, obteve-se cerca de 178 projetos.

A fim de se detectar projetos que tinham a maior taxa de atualização no repositório de análises do Nemo, percebeu-se que o projeto **Apache Maven** era um dos que mais se mantinham atualizados. Feito em linguagem Java, este projeto disponibiliza uma nova análise sobre o código-fonte a cada semana. Diante disso, foi escolhido o Apache Maven como o software a ser monitorado no ambiente de DWing.

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 proposta por Kimball e Ross (2002), apresentada na seção 3.2.1, entende-se que o processo de negócio deste trabalho de conclusão de curso, é avaliar a qualidade do código-fonte continuamente por meio das métricas de código-fonte. Kimball e Ross (2002) enuncia que o segundo passo após a indentificação do processo de negócio, é a identificação da periodicidade dos dados coletados pelo mesmo.

No processo de negócio de métricas de código-fonte, a periodicidade de coleta dos dados é variável, isto é, depende de projeto a projeto. Visando atender a maior parte dos casos possíveis, identificou-se a menor granularidade para agregação: o dia. Isto ocorre, pois as ferramentas de integração contínua, que são muito utilizadas em ambientes ágeis (BECK, 1999a), permitem a estabilização de um pacote de software por dia, logo a menor análise do código-fonte pode ser realizada sobre esse pacote estável diariamente. Contudo, há projetos que liberam versões estáveis com mais tempo, portanto se deve permitir agregações maiores tais como, mês, trimestre, semestre e ano. Para esta etapa do trabalho de conclusão de curso, optou-se pela implementação da agregação diária.

Seguindo os passos subsequentes da metodologia proposta por Kimball e Ross (2002), foram identificados as dimensões e o fato por meio da modelagem multidimensio-

⁴ Disponível em <http://nemo.sonarqube.org/>

nal, tal como mostra a Figura 10, utilizado a ferramenta MySQL Workbench ⁵ que é um software de código aberto com licença GPL 2.0.

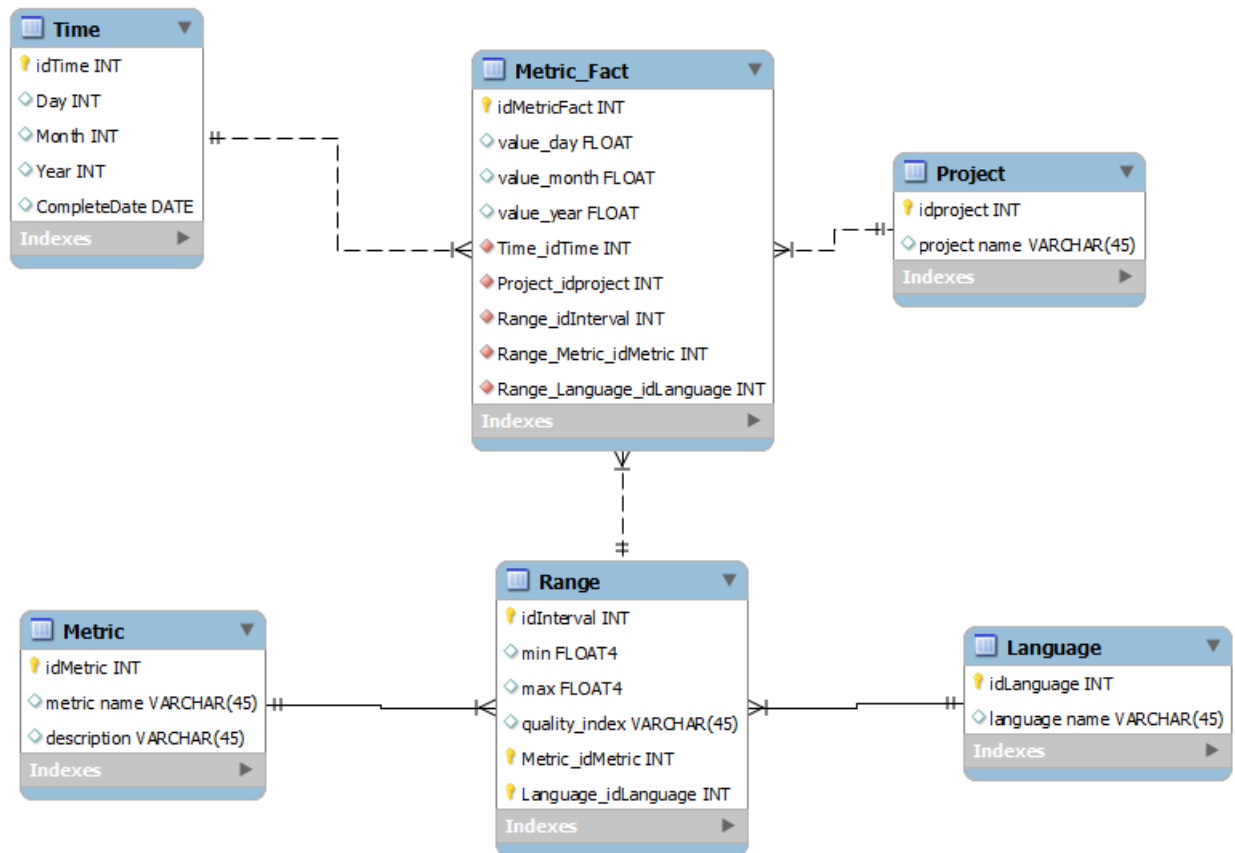


Figura 10 – Modelo multidimensional do *Data Warehouse*

Foi necessário utilizar a normalização na tabela *Range*, para a tabela *Metric* e *Language*, pois cada intervalo de referência, tal como citado na seção 2.3.3, decorrente da análise de Meirelles (2013), refere-se a uma métrica específica e a uma linguagem de programação. O modelo multidimensional, que tem seu SQL resultante apresentado no Apêndice B, foi implementado no MySQL Community Server ⁶ que também possui uma licença de código aberto GPL 2.0

Quanto ao fato (*Metric_Fact*) é possível dizer que o mesmo é não aditivo, pois as métricas de código-fonte não podem ser somadas a nenhuma dimensão. Este fato é calculado por meio do 75º percentil na etapa de transformação dos dados, tal como explicado na seção 2.3.3.

⁵ Disponível em <http://dev.mysql.com/downloads/tools/workbench/>

⁶ Disponível em <http://dev.mysql.com/downloads/mysql/>

4.4 Ferramentas de Dwing

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 Kettle, como é conhecido pela comunidade que o desenvolve, 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 11 e as principais características do Kettle e a análise quanto aos critérios gerais de seleção de ferramentas são apresentadas na Tabela 15.

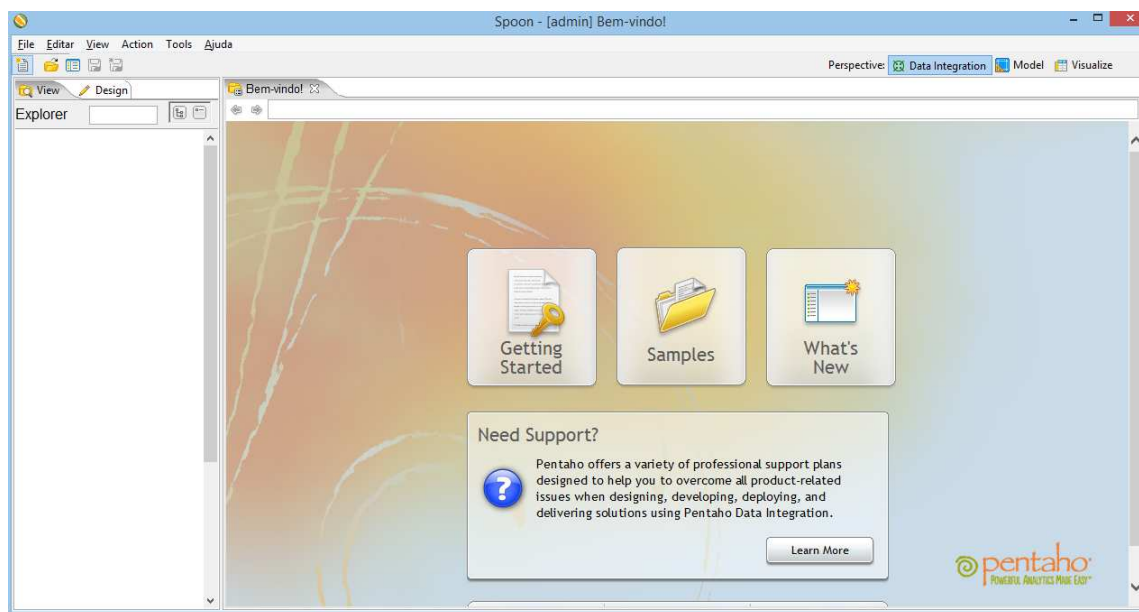


Figura 11 – Interface do Kettle


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 (14/11/2013)	4.4				✓
Quantidade de Commits no Repositório Oficial	10.000+			✓	
Idioma da Documentação	Inglês		✓		
Quantidade de Casos Abertos no <i>Issue Tracker</i>	6000			✓	

Tabela 15 – Características do Kettle e avaliação quanto aos critérios gerais de seleção de ferramentas

O Kettle possui dois tipos de componentes internos: *Transformation* e *Job*. O primeiro permite mover dados de uma origem até um destino, já a *Transformation* 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.

Para a implementação do ETL no Kettle, utilizou-se os arquivos resultantes da análise do SonarQube em JSON e conexão com o MySQL Server, onde foi implementado o Data Warehouse. As implementações detalhadas de *Transformation* e *Job* são apresentados no apêndice A

4.4.2 Implementação das Consultas OLAP e Visualização de Dados

5 Considerações Finais

5.1 Sobre o Trabalho

5.1.1 Resultados Parciais

5.1.2 Trabalhos Futuros

Para trabalho futuros, há i) a investigação da utilização de pesos para cada métrica na análise do código-fonte, possibilitando assim configurações diferenciadas para cada projeto; ii) A implementação do *Dashboard* como componente de visualização; iii) O acompanhamento dos valores obtidos pelas métricas de código-fonte, de forma a determinar a melhor distribuição estatística para as agregações maiores (mês, trimestre, semestre, ano).

Referências

- BASILI, V. R.; CALDIERA, G.; ROMBACH, H. D. *The Goal Question Metric Approach*. [S.l.]: Encyclopedia of Software Engineering, 1996. Citado na página 17.
- BECK, K. Embracing change with extreme programming. *Computer*, v. 32, n. 10, p. 70–77, 1999. ISSN 0018-9162. Citado na página 38.
- BECK, K. *Extreme Programming Explained*. [S.l.]: Addison Wesley, 1999. Citado na página 14.
- BECK, K. *Implementation patterns*. [S.l.]: Pearson Education, 2007. Citado na página 14.
- 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 na página 15.
- 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 na página 15.
- 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 24 e 28.
- 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 21.
- 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 15.
- 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 17.
- 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 27.
- 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 14.

FENTON, N. E.; PFLEEGER, S. L. *Software Metrics: A Rigorous and Practical Approach*. 2 edition. ed. [S.l.]: Course Technology, 1998. 656 p. Citado 2 vezes nas páginas 17 e 18.

FEW, S. *Now you see it: simple visualization techniques for quantitative analysis*. [S.l.]: Analytics Press, 2009. Citado na página 31.

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 na página 15.

FONSECA, R.; SIMOES, A. Alternativas ao xml: Yaml e json. 2007. Citado 3 vezes nas páginas 9, 36 e 37.

FOWLER, M. *Refactoring: improving the design of existing code*. [S.l.]: Addison-Wesley Professional, 1999. Citado na página 14.

GARDNER, S. R. Building the. *Communications of the ACM*, v. 41, n. 9, p. 53, 1998. Citado na página 24.

GOMES, M. M. P. *Métricas e medição no processo de desenvolvimento de software: estudo de caso*. Instituto Universitário de Lisboa: [s.n.], 2011. Citado 2 vezes nas páginas 9 e 17.

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 24.

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 19.

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 21.

INMON, W. H. *Building the Data Warehouse*. New York, NY, USA: John Wiley & Sons, Inc., 1992. ISBN 0471569607. Citado 3 vezes nas páginas 25, 27 e 29.

ISO/IEC 15939. *ISO/IEC 15939: Software Engineering - Software Measurement Process*. [S.l.], 2002. Citado 3 vezes nas páginas 10, 17 e 18.

ISO/IEC 25023. *ISO/IEC 25023: Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - Measurement of system and software product quality*. [S.l.], 2011. Citado 2 vezes nas páginas 9 e 19.

JONES, T. C. *Applied Software Measurement: Assuring Productivity and Quality*. New York: McGraw-Hill, 1991. Citado na página 20.

- KIMBALL, R.; ROSS, M. *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling*. 2nd. ed. New York, NY, USA: John Wiley & Sons, Inc., 2002. ISBN 0471200247, 9780471200246. Citado 5 vezes nas páginas 25, 26, 27, 29 e 38.
- LEHMAN, M. M. Programs, life cycles, and laws of software evolution. *Proc. IEEE*, v. 68, n. 9, p. 1060–1076, September 1980. Citado 2 vezes nas páginas 20 e 37.
- LORENZ, M.; KIDD, J. *Object-Oriented Software Metrics*. [S.l.]: Prentice Hall, 1994. Citado na página 21.
- MCCABE, T. J. A Complexity Measure. *IEEE Transactions Software Engineering*, v. 2, n. 4, p. 308–320, December 1976. Citado na página 20.
- 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 21.
- 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 8 vezes nas páginas 14, 18, 20, 21, 22, 36, 37 e 39.
- 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 18 e 20.
- 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 3 vezes nas páginas 9, 31 e 32.
- NERI, H. R. *Análise, Projeto e Implementação de um Esquema MOLAP de Data Warehouse utilizando SGBD-OR Oracle 8.1*. Universidade Federal da Paraíba - UFPB: [s.n.], 2002. Citado 2 vezes nas páginas 10 e 28.
- 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 14.
- 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 na página 15.
- ROCHA, A. B. *Guardando Históricos de Dimensões em Data Warehouses*. Universidade Federal da Paraíba - Centro de Ciências e Tecnologia: [s.n.], 2000. Citado 6 vezes nas páginas 9, 10, 24, 28, 29 e 30.
- 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 21.

- RUIZ, D. D. A. et al. A data warehousing environment to monitor metrics in software development processes. In: *16th International Workshop on Database and Expert Systems Applications (DEXA 2005), 22-26 August 2005, Copenhagen, Denmark*. [S.l.]: IEEE Computer Society, 2005. p. 936–940. ISBN 0-7695-2424-9. Citado na página 15.
- 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 21.
- 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 21.
- 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 2 vezes nas páginas 15 e 24.
- SOMMERVILLE, I. *Software Engineering*. 9. ed. Harlow, England: Addison-Wesley, 2010. ISBN 978-0-13-703515-1. Citado na página 14.
- TERRA, R.; BIGONHA, R. S. Ferramentas para análise estática de códigos java. Departamento de Ciência da Computação - UFMG, 2008. Citado na página 14.
- 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, 26, 27, 28 e 30.
- WICHMANN, B. et al. Industrial perspective on static analysis. *Software Engineering Journal*, 1995. Citado na página 14.

APÊNDICE A – Implementações no Kettle

APÊNDICE B – SQL do *Data Warehouse*

Abaixo é apresentado o *Data Definition Language* do *Data Warehouse*, para algumas tabelas foram feitos inserts pré-definidos.

```

1 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0; SET
2 @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
3
4 DROP SCHEMA IF EXISTS 'dwing4' ;
5 CREATE SCHEMA IF NOT EXISTS 'dwing4' DEFAULT CHARACTER
6 SET utf8 COLLATE utf8_general_ci ;
7 USE 'dwing4' ;
8
9 -----
10 -- Table 'dwing4`.`Project`
11 -----
12 CREATE TABLE IF NOT EXISTS 'dwing4`.`Project` (
13   'idproject' INT NOT NULL,
14   'project name' VARCHAR(45) NULL,
15   PRIMARY KEY ('idproject'))
16 ENGINE = InnoDB;
17
18
19 -----
20 -- Table 'dwing4`.`Metric`
21 -----
22 CREATE TABLE IF NOT EXISTS 'dwing4`.`Metric` (
23   'idMetric' INT NOT NULL AUTO_INCREMENT,
24   'metric name' VARCHAR(45) NULL,
25   'description' VARCHAR(45) NULL,
26   PRIMARY KEY ('idMetric'))
27 ENGINE = InnoDB;
28
29
30 -----
31 -- Table 'dwing4`.`Time`
32 -----
33 CREATE TABLE IF NOT EXISTS 'dwing4`.`Time` (
34   'idTime' INT NOT NULL AUTO_INCREMENT,
35   'Day' INT NULL,
36   'Month' INT NULL,
37   'Year' INT NULL,

```

```

38     'CompleteDate' DATE NULL,
39     PRIMARY KEY ('idTime'))
40 ENGINE = InnoDB;
41
42
43 -----
44 -- Table 'dwing4'.'Language'
45 -----
46 CREATE TABLE IF NOT EXISTS 'dwing4'.'Language' (
47     'idLanguage' INT NOT NULL AUTO_INCREMENT,
48     'language name' VARCHAR(45) NULL,
49     PRIMARY KEY ('idLanguage'))
50 ENGINE = InnoDB;
51
52
53 -----
54 -- Table 'dwing4'.'Range'
55 -----
56 CREATE TABLE IF NOT EXISTS 'dwing4'.'Range' (
57     'idInterval' INT NOT NULL AUTO_INCREMENT,
58     'min' FLOAT NULL,
59     'max' FLOAT NULL,
60     'quality_index' VARCHAR(45) NULL,
61     'Metric_idMetric' INT NOT NULL,
62     'Language_idLanguage' INT NOT NULL,
63     PRIMARY KEY ('idInterval', 'Metric_idMetric',
64         'Language_idLanguage'),
65     INDEX 'fk_Range_Metric1_idx' ('Metric_idMetric' ASC),
66     INDEX 'fk_Range_Language1_idx' ('Language_idLanguage' ASC),
67     CONSTRAINT 'fk_Range_Metric1'
68         FOREIGN KEY ('Metric_idMetric')
69             REFERENCES 'dwing4'.'Metric' ('idMetric')
70             ON DELETE NO ACTION
71             ON UPDATE NO ACTION,
72     CONSTRAINT 'fk_Range_Language1'
73         FOREIGN KEY ('Language_idLanguage')
74             REFERENCES 'dwing4'.'Language' ('idLanguage')
75             ON DELETE NO ACTION
76             ON UPDATE NO ACTION)
77 ENGINE = InnoDB;
78
79
80 -----
81 -- Table 'dwing4'.'Metric_Fact'
82 -----
83 CREATE TABLE IF NOT EXISTS 'dwing4'.'Metric_Fact' (
84     'idMetricFact' INT NOT NULL AUTO_INCREMENT,

```

```

85  'value_day' FLOAT NULL,
86  'value_month' FLOAT NULL,
87  'value_year' FLOAT NULL,
88  'Time_idTime' INT NOT NULL,
89  'Project_idproject' INT NOT NULL,
90  'Range_idInterval' INT NOT NULL,
91  'Range_Metric_idMetric' INT NOT NULL,
92  'Range_Language_idLanguage' INT NOT NULL,
93  INDEX 'fk_Metric_Fact_Project1_idx' ('Project_idproject' ASC),
94  INDEX 'fk_Metric_Fact_Range1_idx' ('Range_idInterval' ASC,
95    'Range_Metric_idMetric' ASC, 'Range_Language_idLanguage' ASC),
96  PRIMARY KEY ('idMetricFact'),
97  CONSTRAINT 'fk_Metric_Fact_Time1'
98    FOREIGN KEY ('Time_idTime')
99    REFERENCES 'dwing4'.'Time' ('idTime')
100    ON DELETE NO ACTION
101    ON UPDATE NO ACTION,
102  CONSTRAINT 'fk_Metric_Fact_Project1'
103    FOREIGN KEY ('Project_idproject')
104    REFERENCES 'dwing4'.'Project' ('idproject')
105    ON DELETE NO ACTION
106    ON UPDATE NO ACTION,
107  CONSTRAINT 'fk_Metric_Fact_Range1'
108    FOREIGN KEY ('Range_idInterval' , 'Range_Metric_idMetric' ,
109    'Range_Language_idLanguage')
110    REFERENCES 'dwing4'.'Range' ('idInterval' , 'Metric_idMetric' ,
111    'Language_idLanguage')
112    ON DELETE NO ACTION
113    ON UPDATE NO ACTION)
114  ENGINE = InnoDB;
115
116
117  SET SQL_MODE=@OLD_SQL_MODE;
118  SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
119  SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
120
121  -- -----
122  -- Data for table 'dwing4'.'Metric'
123  -- -----
124  START TRANSACTION;
125  USE 'dwing4';
126  INSERT INTO 'dwing4'.'Metric' ('idMetric', 'metric name',
127    'description') VALUES (1, 'LOC', 'Lines of Code');
128  INSERT INTO 'dwing4'.'Metric' ('idMetric', 'metric name',
129    'description') VALUES (2, 'ACCM', 'Average Ciclomatic
130    Complexity Method');
131  INSERT INTO 'dwing4'.'Metric' ('idMetric', 'metric name',

```

```

132     'description') VALUES (3, 'ACC', 'Afferent Conexions per Class');
133 INSERT INTO 'dwing4'.'Metric' ('idMetric', 'metric name',
134     'description') VALUES (4, 'RFC', 'Reponse for Class');
135 INSERT INTO 'dwing4'.'Metric' ('idMetric', 'metric name',
136     'description') VALUES (5, 'LCOM4', 'Lack of Cohesion of Method');
137 INSERT INTO 'dwing4'.'Metric' ('idMetric', 'metric name',
138     'description') VALUES (6, 'NOM', 'Number of Methods');
139 INSERT INTO 'dwing4'.'Metric' ('idMetric', 'metric name',
140     'description') VALUES (7, 'DIT', 'Depth on Tree');
141 INSERT INTO 'dwing4'.'Metric' ('idMetric', 'metric name',
142     'description') VALUES (8, 'NOC', 'Number of Childs');
143
144 COMMIT;
145
146
147 -- -----
148 -- Data for table 'dwing4'.'Language'
149 -- -----
150 START TRANSACTION;
151 USE 'dwing4';
152 INSERT INTO 'dwing4'.'Language' ('idLanguage', 'language name')
153 VALUES (1, 'java');
154 INSERT INTO 'dwing4'.'Language' ('idLanguage', 'language name')
155 VALUES (2, 'c++');
156
157 COMMIT;
158
159
160 -- -----
161 -- Data for table 'dwing4'.'Range'
162 -- -----
163 START TRANSACTION;
164 USE 'dwing4';
165 INSERT INTO 'dwing4'.'Range' ('idInterval', 'min', 'max',
166     'quality_index', 'Metric_idMetric', 'Language_idLanguage')
167 VALUES (1, 0, 33, 'Excelente', 1, 1);
168
169 INSERT INTO 'dwing4'.'Range' ('idInterval', 'min', 'max',
170     'quality_index', 'Metric_idMetric', 'Language_idLanguage')
171 VALUES (2, 34, 87, 'Bom', 1, 1);
172
173 INSERT INTO 'dwing4'.'Range' ('idInterval', 'min', 'max',
174     'quality_index', 'Metric_idMetric', 'Language_idLanguage')
175 VALUES (3, 88, 200, 'Regular', 1, 1);
176
177 INSERT INTO 'dwing4'.'Range' ('idInterval', 'min', 'max',
178     'quality_index', 'Metric_idMetric', 'Language_idLanguage')

```

```
179 VALUES (4, 201, 9999, 'Ruim', 1, 1);
180
181 INSERT INTO 'dwing4'.'Range' ('idInterval', 'min', 'max',
182     'quality_index', 'Metric_idMetric', 'Language_idLanguage')
183 VALUES (5, 0, 31, 'Excelente', 1, 2);
184
185 INSERT INTO 'dwing4'.'Range' ('idInterval', 'min', 'max',
186     'quality_index', 'Metric_idMetric', 'Language_idLanguage')
187 VALUES (6, 32, 84, 'Bom', 1, 2);
188
189 INSERT INTO 'dwing4'.'Range' ('idInterval', 'min', 'max',
190     'quality_index', 'Metric_idMetric', 'Language_idLanguage')
191 VALUES (7, 85, 207, 'Regular', 1, 2);
192
193 INSERT INTO 'dwing4'.'Range' ('idInterval', 'min', 'max',
194     'quality_index', 'Metric_idMetric', 'Language_idLanguage')
195 VALUES (8, 208, 9999, 'Ruim', 1, 2);
196
197 INSERT INTO 'dwing4'.'Range' ('idInterval', 'min', 'max',
198     'quality_index', 'Metric_idMetric', 'Language_idLanguage')
199 VALUES (9, 0, 2.8, 'Excelente', 2, 1);
200
201 INSERT INTO 'dwing4'.'Range' ('idInterval', 'min', 'max',
202     'quality_index', 'Metric_idMetric', 'Language_idLanguage')
203 VALUES (10, 2.9, 4.4, 'Bom', 2, 1);
204
205 INSERT INTO 'dwing4'.'Range' ('idInterval', 'min',
206     'max', 'quality_index', 'Metric_idMetric', 'Language_idLanguage')
207 VALUES (11, 4.5, 6, 'Regular', 2, 1);
208
209 INSERT INTO 'dwing4'.'Range' ('idInterval', 'min', 'max',
210     'quality_index', 'Metric_idMetric', 'Language_idLanguage')
211 VALUES (12, 6.1, 9999, 'Ruim', 2, 1);
212
213 INSERT INTO 'dwing4'.'Range' ('idInterval', 'min', 'max',
214     'quality_index', 'Metric_idMetric', 'Language_idLanguage')
215 VALUES (13, 0, 2.0, 'Excelente', 2, 2);
216
217 INSERT INTO 'dwing4'.'Range' ('idInterval', 'min',
218     'max', 'quality_index', 'Metric_idMetric', 'Language_idLanguage')
219 VALUES (14, 2.1, 4.0, 'Bom', 2, 2);
220
221 INSERT INTO 'dwing4'.'Range' ('idInterval', 'min', 'max',
222     'quality_index', 'Metric_idMetric', 'Language_idLanguage')
223 VALUES (15, 4.1, 6.0, 'Regular', 2, 2);
224
225 INSERT INTO 'dwing4'.'Range' ('idInterval', 'min', 'max',
```

```
226     'quality_index', 'Metric_idMetric', 'Language_idLanguage')
227 VALUES (16, 6.1, 9999, 'Ruim', 2, 2);
228
229 INSERT INTO 'dwing4'.'Range' ('idInterval', 'min', 'max',
230     'quality_index', 'Metric_idMetric', 'Language_idLanguage')
231 VALUES (17, 0, 1, 'Excelente', 3, 1);
232
233 INSERT INTO 'dwing4'.'Range' ('idInterval', 'min', 'max',
234     'quality_index', 'Metric_idMetric', 'Language_idLanguage')
235 VALUES (18, 1.1, 5, 'Bom', 3, 1);
236
237 INSERT INTO 'dwing4'.'Range' ('idInterval', 'min', 'max',
238     'quality_index', 'Metric_idMetric', 'Language_idLanguage')
239 VALUES (19, 5.1, 12, 'Regular', 3, 1);
240
241 INSERT INTO 'dwing4'.'Range' ('idInterval', 'min', 'max',
242     'quality_index', 'Metric_idMetric', 'Language_idLanguage')
243 VALUES (20, 12.1, 9999, 'Ruim', 3, 1);
244
245 INSERT INTO 'dwing4'.'Range' ('idInterval', 'min', 'max',
246     'quality_index', 'Metric_idMetric', 'Language_idLanguage')
247 VALUES (21, 0, 2.0, 'Excelente', 3, 2);
248
249 INSERT INTO 'dwing4'.'Range' ('idInterval', 'min', 'max',
250     'quality_index', 'Metric_idMetric', 'Language_idLanguage')
251 VALUES (22, 2.1, 7, 'Bom', 3, 2);
252
253 INSERT INTO 'dwing4'.'Range' ('idInterval', 'min', 'max',
254     'quality_index', 'Metric_idMetric', 'Language_idLanguage')
255 VALUES (23, 7.1, 15, 'Regular', 3, 2);
256
257 INSERT INTO 'dwing4'.'Range' ('idInterval', 'min', 'max',
258     'quality_index', 'Metric_idMetric', 'Language_idLanguage')
259 VALUES (24, 15.1, 9999, 'Ruim', 3, 2);
260
261 INSERT INTO 'dwing4'.'Range' ('idInterval', 'min', 'max',
262     'quality_index', 'Metric_idMetric', 'Language_idLanguage')
263 VALUES (25, 0, 9, 'Excelente', 4, 1);
264
265 INSERT INTO 'dwing4'.'Range' ('idInterval', 'min', 'max',
266     'quality_index', 'Metric_idMetric', 'Language_idLanguage')
267 VALUES (26, 10, 26, 'Bom', 4, 1);
268
269 INSERT INTO 'dwing4'.'Range' ('idInterval', 'min', 'max',
270     'quality_index', 'Metric_idMetric', 'Language_idLanguage')
271 VALUES (27, 27, 59, 'Regular', 4, 1);
272
```

```
273 INSERT INTO 'dwing4'.'Range' ('idInterval', 'min', 'max',
274   'quality_index', 'Metric_idMetric', 'Language_idLanguage')
275 VALUES (28, 60, 9999, 'Ruim', 4, 1);
276
277 INSERT INTO 'dwing4'.'Range' ('idInterval', 'min', 'max',
278   'quality_index', 'Metric_idMetric', 'Language_idLanguage')
279 VALUES (29, 0, 29, 'Excelente', 4, 2);
280
281 INSERT INTO 'dwing4'.'Range' ('idInterval', 'min', 'max',
282   'quality_index', 'Metric_idMetric', 'Language_idLanguage')
283 VALUES (30, 30, 64, 'Bom', 4, 2);
284
285 INSERT INTO 'dwing4'.'Range' ('idInterval', 'min', 'max',
286   'quality_index', 'Metric_idMetric', 'Language_idLanguage')
287 VALUES (31, 65, 102, 'Regular', 4, 2);
288
289 INSERT INTO 'dwing4'.'Range' ('idInterval', 'min', 'max',
290   'quality_index', 'Metric_idMetric', 'Language_idLanguage')
291 VALUES (32, 103, 9999, 'Ruim', 4, 2);
292
293 INSERT INTO 'dwing4'.'Range' ('idInterval', 'min', 'max',
294   'quality_index', 'Metric_idMetric', 'Language_idLanguage')
295 VALUES (33, 0, 3, 'Excelente', 5, 1);
296
297 INSERT INTO 'dwing4'.'Range' ('idInterval', 'min', 'max',
298   'quality_index', 'Metric_idMetric', 'Language_idLanguage')
299 VALUES (34, 4, 7, 'Bom', 5, 1);
300
301 INSERT INTO 'dwing4'.'Range' ('idInterval', 'min', 'max',
302   'quality_index', 'Metric_idMetric', 'Language_idLanguage')
303 VALUES (35, 8, 12, 'Regular', 5, 1);
304
305 INSERT INTO 'dwing4'.'Range' ('idInterval', 'min', 'max',
306   'quality_index', 'Metric_idMetric', 'Language_idLanguage')
307 VALUES (36, 13, 9999, 'Ruim', 5, 1);
308
309 INSERT INTO 'dwing4'.'Range' ('idInterval', 'min', 'max',
310   'quality_index', 'Metric_idMetric', 'Language_idLanguage')
311 VALUES (37, 0, 5, 'Excelente', 5, 2);
312
313 INSERT INTO 'dwing4'.'Range' ('idInterval', 'min', 'max',
314   'quality_index', 'Metric_idMetric', 'Language_idLanguage')
315 VALUES (38, 6, 10, 'Bom', 5, 2);
316
317 INSERT INTO 'dwing4'.'Range' ('idInterval', 'min', 'max',
318   'quality_index', 'Metric_idMetric', 'Language_idLanguage')
319 VALUES (39, 11, 14, 'Regular', 5, 2);
```

```
320
321 INSERT INTO 'dwing4'.'Range' ('idInterval', 'min', 'max',
322   'quality_index', 'Metric_idMetric', 'Language_idLanguage')
323 VALUES (40, 15, 99999, 'Ruim', 5, 2);
324
325 INSERT INTO 'dwing4'.'Range' ('idInterval', 'min', 'max',
326   'quality_index', 'Metric_idMetric', 'Language_idLanguage')
327 VALUES (41, 0, 8, 'Excelente', 6, 1);
328
329 INSERT INTO 'dwing4'.'Range' ('idInterval', 'min', 'max',
330   'quality_index', 'Metric_idMetric', 'Language_idLanguage')
331 VALUES (42, 9, 17, 'Bom', 6, 1);
332
333 INSERT INTO 'dwing4'.'Range' ('idInterval', 'min', 'max',
334   'quality_index', 'Metric_idMetric', 'Language_idLanguage')
335 VALUES (43, 18, 27, 'Regular', 6, 1);
336
337 INSERT INTO 'dwing4'.'Range' ('idInterval', 'min', 'max',
338   'quality_index', 'Metric_idMetric', 'Language_idLanguage')
339 VALUES (44, 28, 9999, 'Ruim', 6, 1);
340
341 INSERT INTO 'dwing4'.'Range' ('idInterval', 'min', 'max',
342   'quality_index', 'Metric_idMetric', 'Language_idLanguage')
343 VALUES (45, 0, 10, 'Excelente', 6, 2);
344
345 INSERT INTO 'dwing4'.'Range' ('idInterval', 'min', 'max',
346   'quality_index', 'Metric_idMetric', 'Language_idLanguage')
347 VALUES (46, 11, 17, 'Bom', 6, 2);
348
349 INSERT INTO 'dwing4'.'Range' ('idInterval', 'min', 'max',
350   'quality_index', 'Metric_idMetric', 'Language_idLanguage')
351 VALUES (47, 18, 26, 'Regular', 6, 2);
352
353 INSERT INTO 'dwing4'.'Range' ('idInterval', 'min', 'max',
354   'quality_index', 'Metric_idMetric', 'Language_idLanguage')
355 VALUES (48, 27, 9999, 'Ruim', 6, 2);
356
357 INSERT INTO 'dwing4'.'Range' ('idInterval', 'min', 'max',
358   'quality_index', 'Metric_idMetric', 'Language_idLanguage')
359 VALUES (49, 0, 2, 'Excelente', 7, 1);
360
361 INSERT INTO 'dwing4'.'Range' ('idInterval', 'min', 'max',
362   'quality_index', 'Metric_idMetric', 'Language_idLanguage')
363 VALUES (50, 3, 4, 'Bom', 7, 1);
364
365 INSERT INTO 'dwing4'.'Range' ('idInterval', 'min', 'max',
366   'quality_index', 'Metric_idMetric', 'Language_idLanguage')
```



```
367 VALUES (51, 5, 6, 'Regular', 7, 1);
368
369 INSERT INTO 'dwing4'.'Range' ('idInterval', 'min', 'max',
370 'quality_index', 'Metric_idMetric', 'Language_idLanguage')
371 VALUES (52, 7, 9999, 'Ruim', 7, 1);
372
373 INSERT INTO 'dwing4'.'Range' ('idInterval', 'min', 'max',
374 'quality_index', 'Metric_idMetric', 'Language_idLanguage')
375 VALUES (53, 0, 1, 'Excelente', 7, 2);
376
377 INSERT INTO 'dwing4'.'Range' ('idInterval', 'min', 'max',
378 'quality_index', 'Metric_idMetric', 'Language_idLanguage')
379 VALUES (54, 2, 3, 'Bom', 7, 2);
380
381 INSERT INTO 'dwing4'.'Range' ('idInterval', 'min', 'max',
382 'quality_index', 'Metric_idMetric', 'Language_idLanguage')
383 VALUES (55, 3, 4, 'Regular', 7, 2);
384
385 INSERT INTO 'dwing4'.'Range' ('idInterval', 'min', 'max',
386 'quality_index', 'Metric_idMetric', 'Language_idLanguage')
387 VALUES (56, 5, 9999, 'Ruim', 7, 2);
388
389 INSERT INTO 'dwing4'.'Range' ('idInterval', 'min', 'max',
390 'quality_index', 'Metric_idMetric', 'Language_idLanguage')
391 VALUES (57, 0, 1, 'Excelente', 8, 1);
392
393 INSERT INTO 'dwing4'.'Range' ('idInterval', 'min', 'max',
394 'quality_index', 'Metric_idMetric', 'Language_idLanguage')
395 VALUES (58, 1.1, 2, 'Bom', 8, 1);
396
397 INSERT INTO 'dwing4'.'Range' ('idInterval', 'min', 'max',
398 'quality_index', 'Metric_idMetric', 'Language_idLanguage')
399 VALUES (59, 3, 3, 'Regular', 8, 1);
400
401 INSERT INTO 'dwing4'.'Range' ('idInterval', 'min', 'max',
402 'quality_index', 'Metric_idMetric', 'Language_idLanguage')
403 VALUES (60, 4, 9999, 'Ruim', 8, 1);
404
405 INSERT INTO 'dwing4'.'Range' ('idInterval', 'min', 'max',
406 'quality_index', 'Metric_idMetric', 'Language_idLanguage')
407 VALUES (61, 0, 0, 'Excelente', 8, 2);
408
409 INSERT INTO 'dwing4'.'Range' ('idInterval', 'min', 'max',
410 'quality_index', 'Metric_idMetric', 'Language_idLanguage')
411 VALUES (62, 1, 1, 'Bom', 8, 2);
412
413 INSERT INTO 'dwing4'.'Range' ('idInterval', 'min', 'max',
```

```
414     'quality_index', 'Metric_idMetric', 'Language_idLanguage')
415 VALUES (63, 1, 2, 'Regular', 8, 2);
416
417 INSERT INTO 'dwing4'.'Range' ('idInterval', 'min', 'max',
418
419     'quality_index', 'Metric_idMetric', 'Language_idLanguage')
420 VALUES (64, 3, 9999, 'Ruim', 8, 2);
421
422 COMMIT;
```
