



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-
38 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

*‘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, antecidas 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)	16
Figura 2 – Modelo de Qualidade do Produto da ISO 25023 adaptado de ISO/IEC 25023 (2011)	18
Figura 3 – Arquitetura de um ambiente de <i>Data Warehousing</i> extraído de Rocha (2000)	23
Figura 4 – Exemplo de Esquema Estrela extraído de Times (2012)	25
Figura 5 – Exemplo de Cubo Multidimensional de dados extraído de Times (2012)	26
Figura 6 – Arquitetura do Ambiente de Dwing para Métricas de Código-Fonte . . .	30
Figura 7 – Ciclos de Desenvolvimento do Trabalho de Conclusão de Curso	33
Figura 8 – Gráfico de <i>Gantt</i> do Trabalho de Conclusão de Curso	33
Figura 9 – Quadro <i>Kanban</i> do Trabalho de Conclusão de Curso	34

Lista de tabelas

Tabela 1 – Modelo de Informação para metodologias ágeis com base na ISO/IEC 15939 (2002)	17
Tabela 2 – Nome dos Intervalos de Frequência	21
Tabela 3 – Intervalos das Métricas para Java e C++	22
Tabela 4 – Diferenças entre OLAP e OLTP extraído de Times (2012), Rocha (2000) e Neri (2002)	27
Tabela 5 – Exemplo do Total de Vendas de uma Rede de Lojas no mês de Novembro	28
Tabela 6 – Exemplo do Total de Vendas de uma rede de lojas no mês de novembro com a dimensão Produto	28
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	29
Tabela 9 – Critérios Gerais de escolha das ferramentas	31
Tabela 10 – Critérios Específicos para Ferramenta de Análise Estática de Código-Fonte	31
Tabela 11 – Características do SonarQube e do Analizo	32

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	13
1.1	Objetivos	14
1.1.1	Objetivo Geral	14
1.1.2	Objetivos Específicos	15
1.2	Metodologia de Pesquisa	15
1.3	Organização do Trabalho	15
2	Métricas de Software	16
2.1	Processo de Medição de Software	16
2.2	Classificação das Métricas de Software	17
2.3	Métricas de Código-Fonte	18
2.3.1	Métrica de Tamanho e Complexidade	19
2.3.2	Métricas de Orientação à Objetos	19
2.3.3	Intervalos da Métricas	20
3	Data Warehousing (DWing)	23
3.1	<i>Extraction-Transformation-Load</i> (ETL)	24
3.2	<i>Data Warehouse</i>	24
3.2.1	Metodologia do Projeto do <i>Data Warehouse</i>	26
3.3	<i>On-Line Analytical Processing</i> (OLAP)	27
3.4	Visualização de Dados	29
4	Implementação do Ambiente de DWing	30
4.1	Arquitetura da Implementação	30
4.2	Ferramenta de Análise Estática de Código-Fonte	31
4.3	Ferramenta de Extração, Transformação e Carga (ETL)	32
4.4	Projeto do <i>Data Warehouse</i>	32
4.5	Ferramenta de Apoio as Consultas OLAP e Visualização de Dados	32
5	Considerações Finais	33
5.1	Sobre o Trabalho	33
5.1.1	Cronograma	33
5.1.2	Considerações Finais	34
5.1.3	Trabalhos Futuros	34
	Referências	35

1 Introdução

As metodologias ágeis vem ganhando cada vez mais espaço no mercado global de desenvolvimento de software, pois enfatizam a qualidade do produto sobre a qualidade do processo, procurando minimizar a execução de atividades não essenciais ao longo do ciclo de vida de desenvolvimento de software.

O eXtreme Programming (XP), que é uma metodologia ágil, tem a codificação como atividade chave durante um projeto de desenvolvimento de software (BECK, 1999). Isso se torna perceptível quando se analisa algumas práticas do XP,¹ tais como:

- 1 - Padronização do Código: O código é a principal forma de comunicação entre a equipe. A padronização de código o torna consistente e fácil para leitura e refatoração por todo o time.
- 2 - Propriedade Coletiva do Código: Cada programador pode melhorar qualquer parte do código quando houver a oportunidade.
- 3 - Programação em Pares: Todo código é escrito por duas pessoas: uma que olha para uma máquina, e outra com um teclado e um mouse.
- 4 - Integração Contínua: Todo novo código é integrado ao sistema e, quando integrado, o sistema é totalmente reconstruído do zero e todos os testes devem passar. Caso contrário, o código é descartado.

Dada a importância do código-fonte, infere-se a qualidade do trabalho produzido pela qualidade do código-fonte, sendo um dos métodos mais utilizados para tal a análise estática de código-fonte. Durante anos, vários trabalhos foram publicados visando a definição formal de métodos de análise estática de código-fonte. Vide os trabalhos de Wichmann et al. (1995), Nielson, Nielson e Hankin (1999), Emanuelsson e Nilsson (2008). Contudo as ferramentas que realizam o procedimento no código-fonte ainda apresentam 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.

¹ Documentação disponível em <http://www.extremeprogramming.org>

- 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 acima 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 código-fonte.

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 *Dwing*, para extrair e visualizar métricas de código-fonte afim

de a interpretação das métricas de código-fonte possa ser utilizada de forma a identificar a qualidade interna do produto.

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 múltiplos casos de avaliação de qualidade de código-fonte de softwares livres.

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, as métricas de software bem como as métricas de código-fonte. O Capítulo 3 apresenta conceitualmente o ambiente de *data warehousing* (DWing). O Capítulo 4 apresenta a proposta de 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

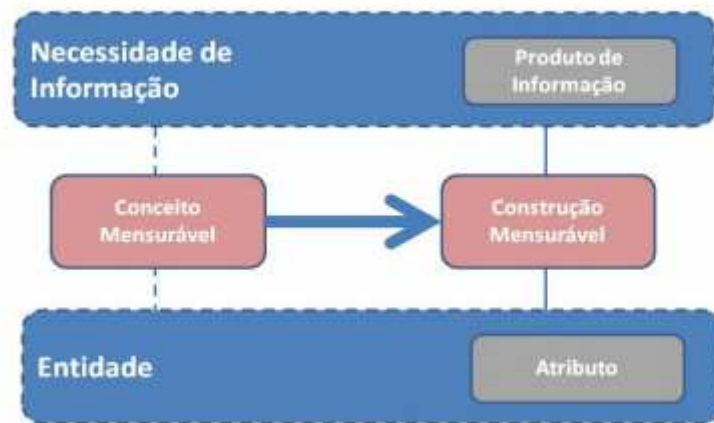


Figura 1 – Modelo de Informação da ISO 15939 extraído de [Gomes \(2011\)](#)

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:

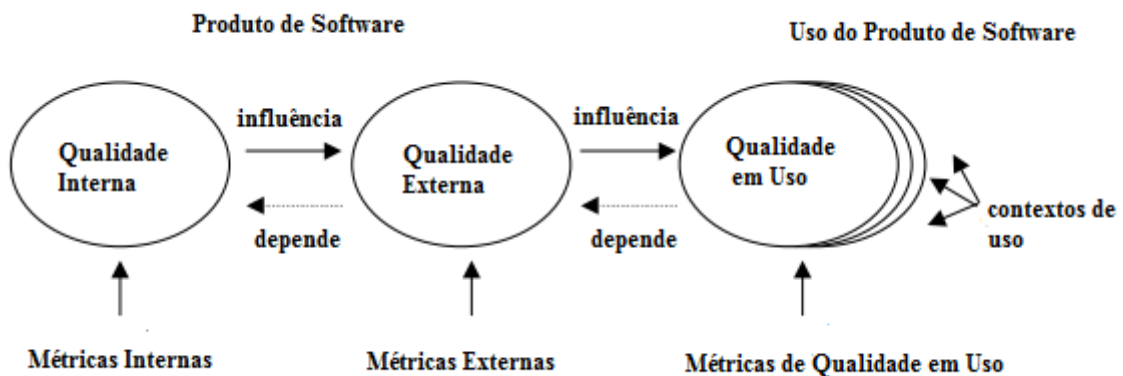


Figura 2 – Modelo de Qualidade do Produto da ISO 25023 adaptado de ISO/IEC 25023 (2011)

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). Dentre as inúmeras características que fazem um bom software, várias delas podem ser percebidas no código-fonte e algumas são exclusivas dele (MEIRELLES, 2013), isto é, o código-fonte revela muitas características de um bom desenvolvimento de software.

Uma das formas de verificação da qualidade do código fonte é a análise estática de código, que se refere à uma análise automatizada das estruturas internas do código, em nível de linguagem de programação, sem realizar a execução, afim de obter métricas de código-fonte (TERRA; BIGONHA, 2008) (EMANUELSSON; NILSSON, 2008) (WICHMANN et al., 1995) (NIELSON; NIELSON; HANKIN, 1999) (SOMMERVILLE, 2010).

As métricas de código-fonte são 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 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, e partir destas abstrair 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 coesa (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).

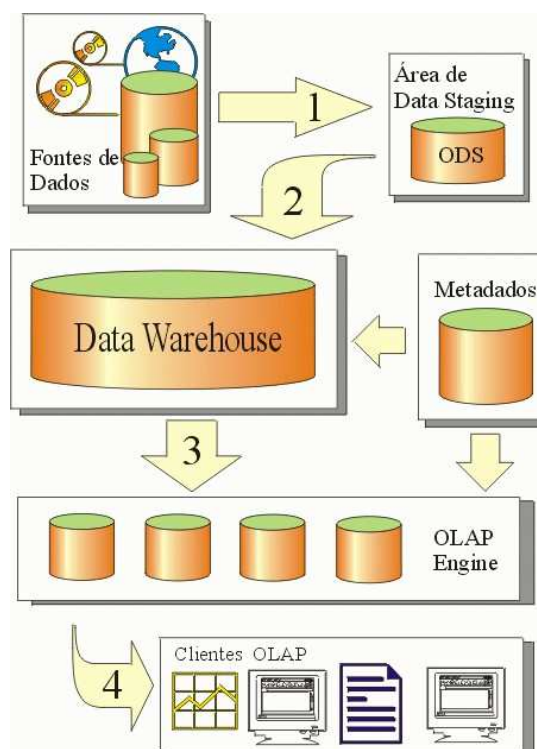


Figura 3 – Arquitetura de um ambiente de *Data Warehousing* extraído de Rocha (2000)

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

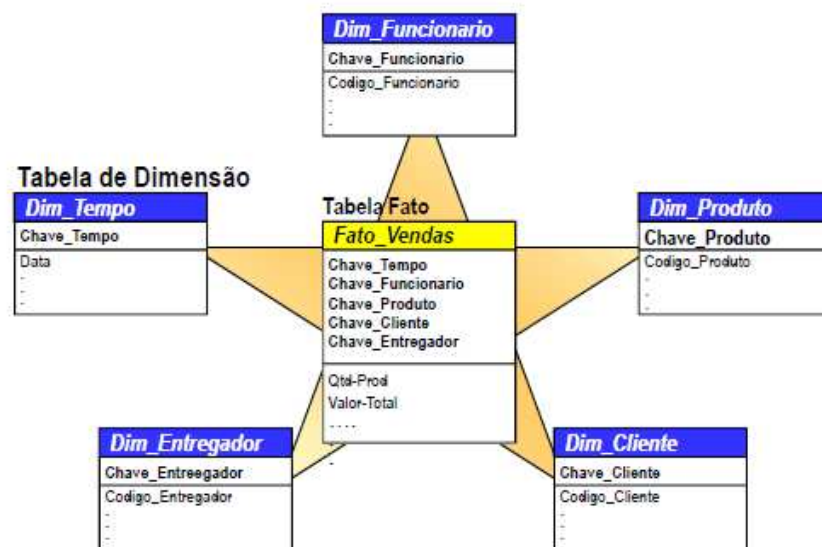


Figura 4 – Exemplo de Esquema Estrela extraído de Times (2012)



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) Declarar a granularidade dos dados necessários para processo de negócio, isto é, verificar a periodicidade de coleta dos dados (diários, semanais, mensais, trimestrais, semestrais ou anuais);
- 3) Escolher as dimensões;
- 4) Identificar os fatos;

Seguindo a metodologia proposta por [Kimball e Ross \(2002\)](#), entende-se que o processo de negócio, neste trabalho de conclusão de curso, foi avaliar a qualidade do código-fonte continuamente por meio das métricas de código-fonte. Os demais passos da metodologia, tais como, a granularidade dos dados, identificação dos fatos e dimensões serão apresentados no Capítulo 4.

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.

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

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 8, obtendo assim a Tabela ??

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

- *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 ?? para Tabela ??

3.4 Visualização de Dados

Os números tem importantes histórias a contar e cabe a quem desejar comunicá-los, saber a forma mais efetiva de fazê-lo.

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 6.

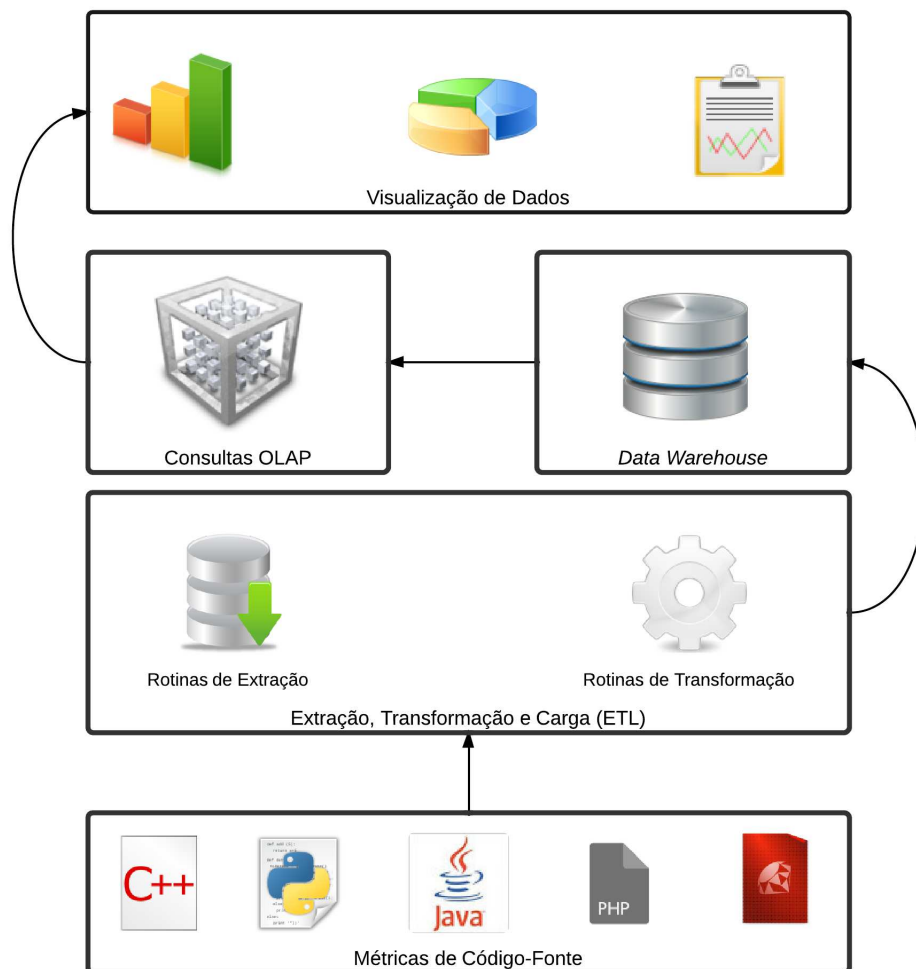


Figura 6 – Arquitetura do Ambiente de Dwing para Métricas de Código-Fonte

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 9.

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

Tabela 9 – 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 10

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 10 – Critérios Específicos para Ferramenta de Análise Estática de Código-Fonte

Após a realização de uma busca por ferramentas, foram encontradas o SonarQube¹ e Analizo² que as principais características de ambas são apresentadas na Tabela 11

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 ??

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

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

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



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	CSV, YAML	JSON, XML, CSV
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> : 7532	<i>Commits</i> : 639
	<i>Branches</i> : 16	<i>Branches</i> : 1
	Contribuidores: 18	Contribuidores: 7
	<i>Releases</i> : 36	<i>Releases</i> : 27
Número de Casos Abertos no <i>Issue Tracker</i> em 14/11/2013	552	26
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	1.17.0	4.0
Data de Lançamento da Última Versão Estável	31 de Janeiro de 2013	4 de Novembro de 2013

Tabela 11 – Características do SonarQube e do Analizo

4.3 Ferramenta de Extração, Transformação e Carga (ETL)

4.4 Projeto do *Data Warehouse*

4.5 Ferramenta de Apoio as Consultas OLAP e Visualização de Dados

5 Considerações Finais

5.1 Sobre o Trabalho

5.1.1 Cronograma

O Trabalho de Conclusão de Curso foi planejado, em longo prazo, em quatro ciclos de desenvolvimento de vinte e um dias cada um. As figuras 7 e 8, que foram extraídas da ferramenta OpenProj¹, apresentam o cronograma geral e o gráfico de Gantt do presente trabalho.

		Nome	Duração	Início	Término	Predecessoras
1		Preparação do TCC	21 dias	22/07/13 08:00	19/08/13 17:00	
2		Proposta do TCC	0 dias	19/08/13 08:00	19/08/13 08:00	1
3		Primeiro Ciclo do TCC	22 dias	19/08/13 08:00	17/09/13 17:00	2
4		Primeira Versão do TCC	0 dias	17/09/13 17:00	17/09/13 17:00	3
5		Segundo Ciclo do TCC	21 dias	18/09/13 08:00	16/10/13 17:00	4
6		Segunda Versão do TCC	0 dias	16/10/13 17:00	16/10/13 17:00	5
7		Terceiro Ciclo	21 dias	17/10/13 08:00	14/11/13 17:00	6
8		Terceira Versão do TCC	0 dias	14/11/13 17:00	14/11/13 17:00	7
9		Quarto Ciclo do TCC	22 dias	15/11/13 08:00	16/12/13 17:00	8
10		Defesa do TCC	0 dias	16/12/13 17:00	16/12/13 17:00	9

Figura 7 – Ciclos de Desenvolvimento do Trabalho de Conclusão de Curso

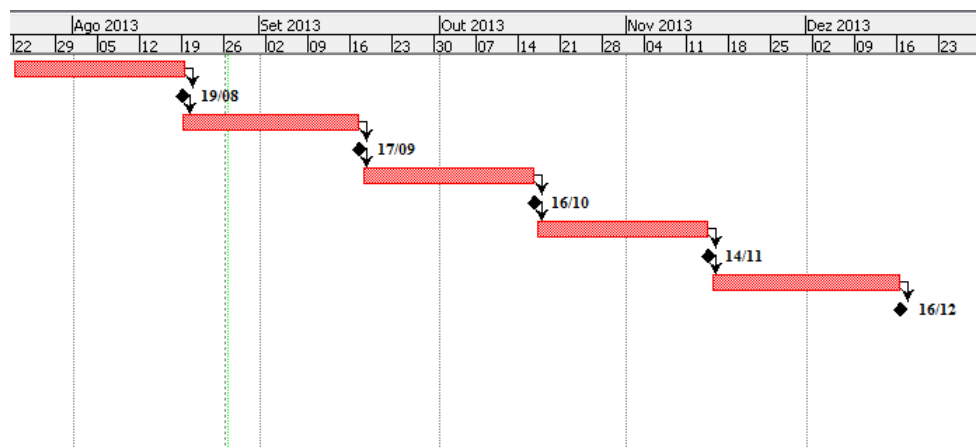


Figura 8 – Gráfico de Gantt do Trabalho de Conclusão de Curso

Para cada ciclo de desenvolvimento do Trabalho de Conclusão de Curso, utilizou-se o Quadro *Kanban* para controle das atividades. O termo *kanban* vem do japonês e significa literalmente "cartão de sinalização". O quadro *kanban*, surgiu na Toyota, como um meio visual para controlar o fluxo da produção, limitando o tamanho do trabalho em progresso (MOURA, 1999).

¹ Documentação e download disponível em <http://sourceforge.net/projects/openproj/>

Posteriormente, o método foi incorporado no processo de desenvolvimento de software pela metodologia ágil, *Scrum* (SCHWABER, 2004). O quadro *Kanban* que foi utilizado no presente trabalho, foi criado na ferramenta *online*, *Kanban Flow*², e foi dividido em três colunas: **Backlog**, que são tarefas a fazer, **Em Progresso**, que são tarefas iniciadas e **Concluído** que mostra as tarefas que já foram executadas. A figura 9 mostra o *Kanban* ao final do segundo ciclo de desenvolvimento.

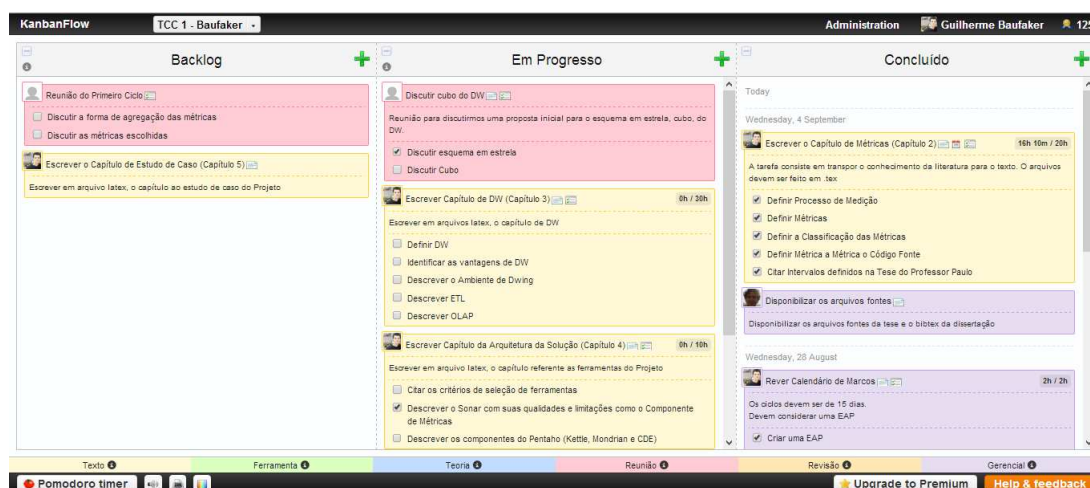


Figura 9 – Quadro *Kanban* do Trabalho de Conclusão de Curso

5.1.2 Considerações Finais

5.1.3 Trabalhos Futuros

² Disponível em <https://kanbanflow.com>

Referências

BASILI, V. R.; CALDIERA, G.; ROMBACH, H. D. *The Goal Question Metric Approach*. [S.l.]: Encyclopedia of Software Engineering, 1996. Citado na página 16.

BECK, K. *Extreme Programming Explained*. [S.l.]: Addison Wesley, 1999. Citado na página 13.

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 14.

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 14.

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 23 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 20.

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 14.

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 16.

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 2 vezes nas páginas 13 e 19.

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 16 e 17.

- 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 14.
- GARDNER, S. R. Building the. *Communications of the ACM*, v. 41, n. 9, p. 53, 1998. Citado na página 23.
- 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 16.
- 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 23.
- 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 18.
- 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 20.
- INMON, W. H. *Building the Data Warehouse*. New York, NY, USA: John Wiley & Sons, Inc., 1992. ISBN 0471569607. Citado 3 vezes nas páginas 24, 26 e 28.
- ISO/IEC 15939. *ISO/IEC 15939: Software Engineering - Software Measurement Process*. [S.l.], 2002. Citado 3 vezes nas páginas 10, 16 e 17.
- ISO/IEC 25023. *ISO/IEC 25023: Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - Measurement of system and software product quality*. [S.l.], 2011. Citado 2 vezes nas páginas 9 e 18.
- JONES, T. C. *Applied Software Measurement: Assuring Productivity and Quality*. New York: McGraw-Hill, 1991. Citado na página 19.
- 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 24, 25, 26, 27 e 28.
- 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 19.
- LORENZ, M.; KIDD, J. *Object-Oriented Software Metrics*. [S.l.]: Prentice Hall, 1994. Citado na página 20.
- MCCABE, T. J. A Complexity Measure. *IEEE Transactions Software Engineering*, v. 2, n. 4, p. 308–320, December 1976. Citado na página 19.

- 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 20.
- 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 6 vezes nas páginas 14, 17, 18, 19, 20 e 21.
- 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 17 e 19.
- MOURA, R. *Kanban: a simplicidade do controle da produção*. [S.l.]: IMAM, 1999. Citado na página 33.
- 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 27.
- NIELSON, F.; NIELSON, H. R.; HANKIN, C. *Principles of Program Analysis*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1999. ISBN 3540654100. Citado 2 vezes nas páginas 13 e 19.
- 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 14.
- 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, 23, 27, 28 e 29.
- 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 20.
- 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 14.
- SCHWABER, K. *Agile Project Management With Scrum*. Redmond, WA, USA: Microsoft Press, 2004. ISBN 073561993X. Citado na página 34.
- 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 20.
- 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 20.

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 14 e 23.

SOMMERVILLE, I. *Software Engineering*. 9. ed. Harlow, England: Addison-Wesley, 2010. ISBN 978-0-13-703515-1. Citado na página 19.

TERRA, R.; BIGONHA, R. S. Ferramentas para análise estática de códigos java. Departamento de Ciência da Computação - UFMG, 2008. Citado na página 19.

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, 25, 26, 27 e 29.

WICHMANN, B. et al. Industrial perspective on static analysis. *Software Engineering Journal*, 1995. Citado 2 vezes nas páginas 13 e 19.