



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

# **Extração e Visualização de métricas de código-fonte em um ambiente de *Dwing*: Do Sonar ao Pentaho**

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 *Dwing*: Do Sonar ao Pentaho**

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 *Dwing*:  
Do Sonar ao Pentaho/ Guilherme Baufaker Rêgo. – Brasília, DF, 2013-  
31 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. Palavra-chave01. 2. Palavra-chave02. I. Prof. Msc. Hilmer Rodrigues Neri.  
II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Extração e Visuali-  
zação de métricas de código-fonte em um ambiente de *Dwing*: Do Sonar ao Pentaho

CDU 02:141:005.6

---

Guilherme Baufaker Rêgo

# **Extração e Visualização de métricas de código-fonte em um ambiente de *Dwing*: Do Sonar ao Pentaho**

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, 01 de junho de 2013:

---

**Prof. Msc. Hilmer Rodrigues Neri**  
Orientador

---

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

---

**Titulação e Nome do Professor**  
**Convidado 01**  
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) . . . . .	17
Figura 2 – Modelo de Qualidade do Produto da ISO 25023 adaptado de ISO/IEC 25023 (2011) . . . . .	18
Figura 3 – Ciclos de Desenvolvimento do Trabalho de Conclusão de Curso . . . . .	27
Figura 4 – Gráfico de <i>Gantt</i> do Trabalho de Conclusão de Curso . . . . .	27
Figura 5 – Quadro <i>Kanban</i> do Trabalho de Conclusão de Curso . . . . .	28

# Lista de tabelas

Tabela 1	– Nome dos Intervalos de Frequência . . . . .	21
Tabela 2	– Intervalos das Métricas para Java e C++ . . . . .	22

# 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>
PDCA	<i>Plan-Do-Check-Act</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

<b>1</b>	<b>Introdução</b>	<b>13</b>
1.1	Objetivos	14
1.1.1	Objetivos Gerais	14
1.1.2	Objetivos Específicos	15
1.1.3	Metodologia de Pesquisa	15
1.1.4	Organização do Trabalho	15
<b>2</b>	<b>Métricas de Software</b>	<b>16</b>
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	21
<b>3</b>	<b>DWing</b>	<b>23</b>
3.1	Data Warehouse	23
3.2	Arquitetura de DWing	23
3.3	Projeto de DW	23
<b>4</b>	<b>Solução</b>	<b>24</b>
4.1	Arquitetura da Solução	24
4.2	SonarQube	24
4.2.1	Escolha do SonarQube	24
4.2.2	Limitações Observadas	25
4.3	Pentaho	26
<b>5</b>	<b>Considerações Finais</b>	<b>27</b>
5.1	Sobre o Trabalho	27
5.1.1	Cronograma	27
	<b>Referências</b>	<b>29</b>

# 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,<sup>1</sup> 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, Funções) do código-fonte.
- P2 - Ausência de mecanismos de tratamento, separação, recuperação, organização e persistência de dados.

---

<sup>1</sup> 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 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. Além disso, se os dados não são representativos, qualquer análise fica a cargo de experiências anteriores com as métricas de 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 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 suprem os problemas encontrados nas ferramentas de análise estática de código-fonte.**

## 1.1 Objetivos

Esta seção apresenta os objetivos gerais e específicos deste Trabalho de Conclusão de Curso.

### 1.1.1 Objetivos Gerais

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.

### 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 em uma estrutura de estudo de caso.
- OE3 - Incorporar indicadores qualitativos para cada métrica de código-fonte.
- OE4 - Facilitar o entendimento das métricas de código-fonte.
- OE5 - Apresentar, em formas visuais, as análises obtidas das métricas de código-fonte.

### 1.1.3 Metodologia de Pesquisa

### 1.1.4 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 *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

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. (FENTON; PFLEEGER, 1998). Visando 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.<sup>1</sup>

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.

Uma construção mensurável tem em sua origem um conceito mensurável tais como 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.

---

<sup>1</sup> A definição formal da ISO/IEC 15939 (2002) não utiliza o termo métrica, sendo que este é definido pelo 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 presente trabalho

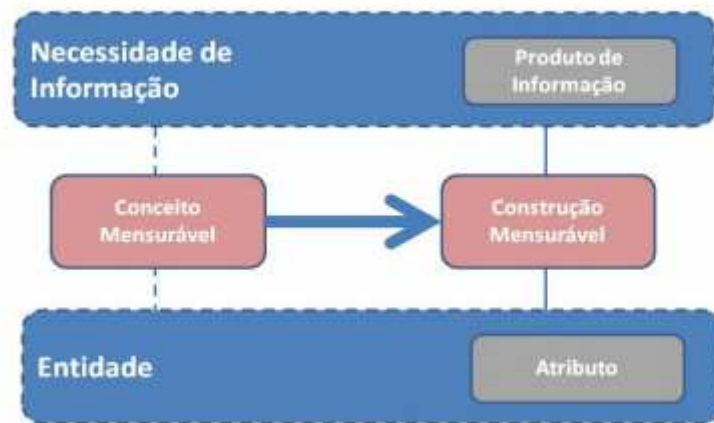


Figura 1 – Modelo de Informação da ISO 15939 extraído de [Gomes \(2011\)](#)

Em metodologias ágeis, a principal necessidade de informação técnica é quanto a qualidade, conceito mensurável, do código-fonte, entidade, por meio de seus elementos internos, atributos da entidade.

## 2.2 Classificação das Métricas de Software

As métricas 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:

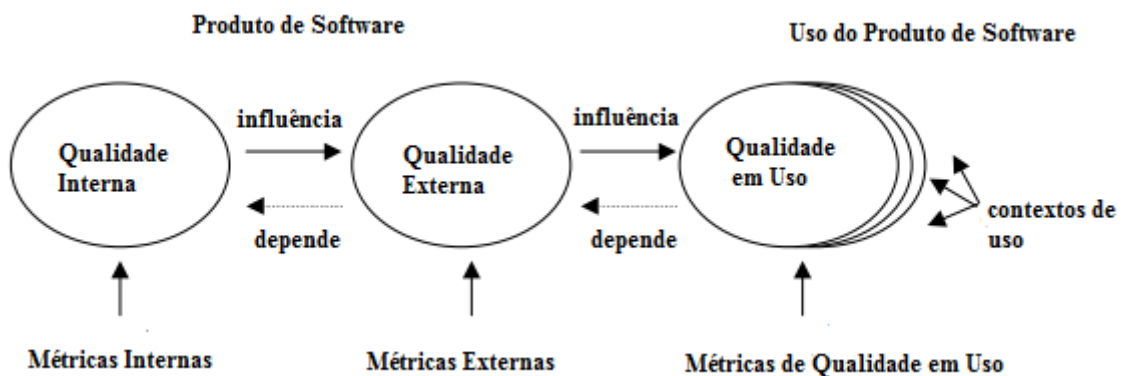


Figura 2 – Modelo de Qualidade do Produto da ISO 25023 adaptado de ISO/IEC 25023 (2011)

**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

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

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, ).

Popularmente, o termo análise estática de código 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 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 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 Ciclomática por Método** mede a complexidade dos métodos ou funções de um programa. Essa métrica pode ser representada através de um grafo de fluxo de controle (MCCABE, 1976). O uso de estruturas de controle, tais como, *if*, *else*, *while* aumentam a complexidade ciclomática de um 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) classifica estes intervalos, em *muito frequente, frequente, pouco frequente e não frequente*. Visando o o melhor entendimento das métricas, resolveu-se renomear tal como a Tabela 1. Posteriormente, é apresentada a Tabela 2, 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 1 – 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 2 – Intervalos das Métricas para Java e C++

Os intervalos, que foram apresentados na Tabela 2 serão utilizados como indicadores de qualidade de código-fonte, de modo a simplificar a interpretação da qualidade do código-fonte nos *Dashboards*, do ambiente de *Dwing*, a ser apresentado a seguir.

## 3 DWing

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

### 3.1 Data Warehouse

A definição mais difundida entre os pesquisadores apresenta o *Data Warehouse* (DW) com 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). Em uma outra definição de *Data Warehouse* (DW) é um repositório central e consolidado (KIMBALL; ROSS, 2002)

80 O DW é um repositório de dados central e consolidado

### 3.2 Arquitetura de DWing

### 3.3 Projeto de DW



## 4 Solução

### 4.1 Arquitetura da Solução

### 4.2 SonarQube

O SonarQube <sup>1</sup>, anteriormente conhecido como Sonar, é uma ferramenta que realiza coleta de dados do código-fonte. Em sua documentação, são citados sete eixos de controle de qualidade cobertos na ferramenta:

- 1 - Duplicação de Código
- 2 - Detecção de Complexidade de Código
- 3 - Padrão de Codificação
- 4 - Testes Unitários
- 5 - Arquitetura e Projeto
- 6 - Potenciais Defeitos
- 7 - Comentários

#### 4.2.1 Escolha do SonarQube

O SonarQube foi escolhido como ferramenta para extração de dados do código-fonte devido as seguintes características:

**Ferramenta Livre:** O SonarQube é uma ferramenta livre que está licenciada sob a GNU LPGL 3, que é uma das licenças de software livre.

**Suporte a várias Linguagens de Programação:** É realizada coleta em mais de 20 linguagens de programação, como por exemplo, Java, C#, Python, C++, .Net, PHP e entre outras.

**Multiplataforma:** Há suporte para os sistemas operacionais Windows, Linux, Mac OS e há também para servidores de aplicação Java, como por exemplo, o Apache Tomcat.

---

<sup>1</sup> Toda a documentação e *downloads* estão disponíveis em <http://www.sonarqube.org/>

**Integração com Ferramentas de Integração Contínua:** é possível realizar a integração com ferramentas de integração contínua, como por exemplo, Jenkins. Assim a cada versão bem sucedida de construção do código-fonte, é possível aferir a qualidade do código.

**Persistência em Base de Dados:** toda a coleta de dados é persistida em uma base de dados. Há suporte para as principais ferramentas de bancos de dados utilizadas no mercado: MySQL, PostgreSQL e Oracle.

**Interface Amigável:** Diferentemente de outras ferramentas disponíveis no mercado, o SonarQube tem interface web, ou seja, é possível visualização das métricas do código-fonte seja realizada em navegador a escolha do usuário. É possível ainda visualizar todo o código-fonte analisado.

**Integração com Ferramentas de Gestão de Defeitos:** É possível integrar com ferramentas de gestão de defeitos, tais como Mantis e JIRA.

**Suporte a Muti produtos:** Há suporte para visualização de métricas de mais um Produto.

**Suporte a Multi-Idiomas:** Há suporte para Inglês, Japonês, Russo, Francês, Italiano, Espanhol e Português.

**Diversidade de Plugins:** O SonarQube permite a extensão de funcionalidades por meio de plugins, sendo que há uma variedade de plugins para mais diversas necessidades disponíveis para *download*. Cabe ressaltar que alguns são gratuitos e outros tem licença comercial.

**Presença de Web Service:** Há um Web Service que extrai as métricas disponíveis para diversos formatos, tais como JSON, XML e CSV.

#### 4.2.2 Limitações Observadas

Após uma análise ostensiva na ferramenta, concluiu-se que o SonarQube apresenta as seguintes limitações:

**Ausência de Visão Consolidada de Métricas:** A maior parte das métricas que SonarQube estão apenas no âmbito de Classes ou Pacotes.

**Impossibilidade Definição de Múltiplos Intervalos:** O sonarQube não permite a configuração de múltiplos intervalos. É definido apenas, um intervalo de alerta (amarelo) e um intervalo de atenção (vermelho).

**Ausência de Avaliações Qualitativas:** As métricas não trazem nenhuma avaliação qualitativa associada, ou seja, sem o conhecimento específico do significado de cada métrica, a interpretação dos valores obtidos não é representativo nem elucidativo sobre a qualidade do código.

**Métricas como Violações:** O SonarQube utiliza para coletar dados de código-fonte, em algumas linguagens de programação, coletores que são identificadores de padrões de codificação. Isso quer dizer que algumas métricas não são apresentadas com valor absoluto, apenas são indicadas como violações à regras estabelecidas pelos padrões.

## 4.3 Pentaho

O Pentaho é um software de código aberto para inteligência empresarial, desenvolvido em Java pela Pentaho Corporation, que apresenta soluções cobrindo as áreas de ETL, *reporting*, OLAP e mineração de dados (*data-mining*).

## 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 3 e 4, que foram extraídas da ferramenta OpenProj<sup>1</sup>, 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 3 – Ciclos de Desenvolvimento do Trabalho de Conclusão de Curso

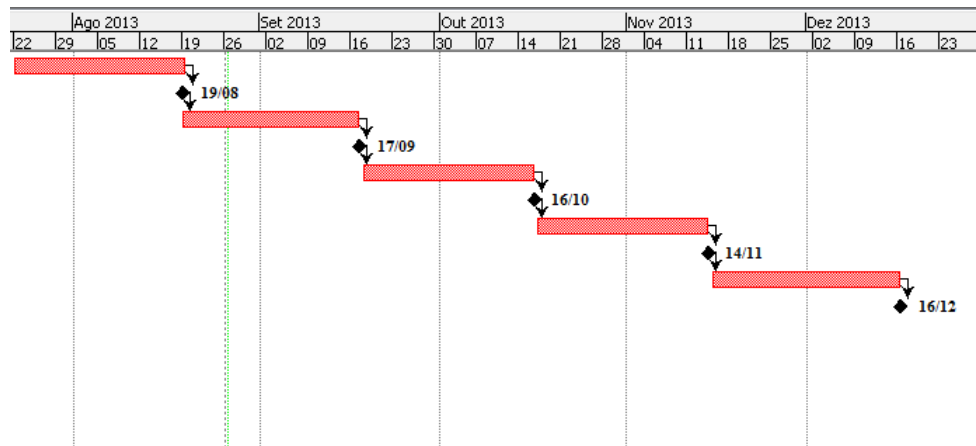


Figura 4 – Gráfico de *Gantt* do Trabalho de Conclusão de Curso

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).

<sup>1</sup> 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*<sup>2</sup>, 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 5 mostra o *Kanban* ao final do segundo ciclo de desenvolvimento.

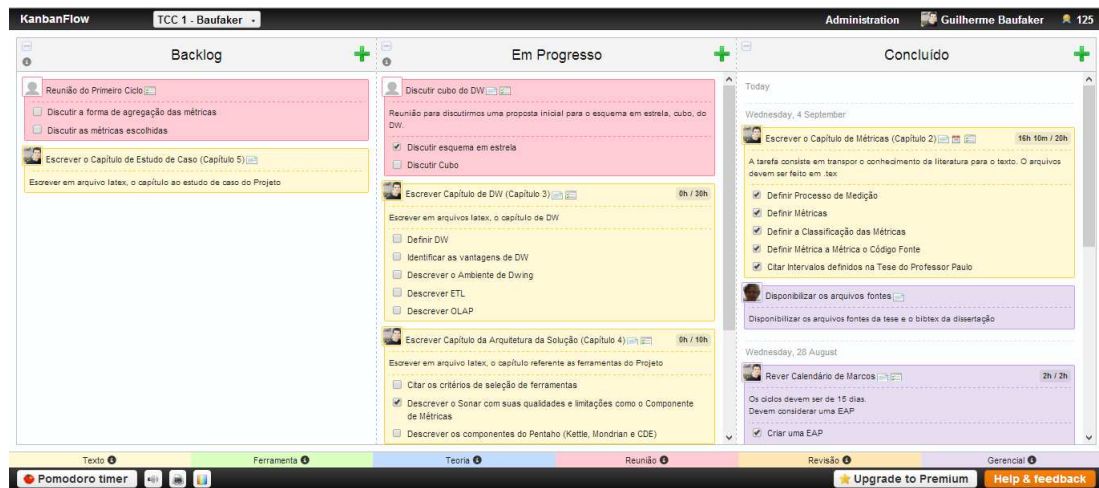


Figura 5 – Quadro *Kanban* do Trabalho de Conclusão de Curso

<sup>2</sup> 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.
- 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.
- 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.

GOMES, M. M. P. *Métricas e medição no processo de desenvolvimento de software: estudo de caso*. 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 23.

HARMAN, M. *1 Why Source Code Analysis and Manipulation Will Always Be Important*. 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 20.

INMON, W. H. *Building the Data Warehouse*. New York, NY, USA: John Wiley & Sons, Inc., 1992. ISBN 0471569607. Citado na página 23.

ISO/IEC 15939. *ISO/IEC 15939: Software Engineering - Software Measurement Process*. [S.l.], 2002. Citado 2 vezes nas páginas 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 na página 23.

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 5 vezes nas páginas 14, 17, 18, 19 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 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.

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 28.

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.

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