



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-
29 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

Agradeço aos meus orientadores Prof. Hilmer e Prof. Paulo por todo o apoio concedido durante o Trabalho de Conclusão de Curso. Agradeço aos meus pais por terem lutado todos os dias e incansavelmente para mostrar que a educação é dos únicos bens duráveis na vida. Agradeço à companheira Luisa Helena Lemos da Cruz por todo o apoio concedido durante os dias difíceis que atendeu a construção do presente trabalho. Agradeço a Prof. Eneida Gonzales por ter me incentivado a enfrentar os desafios acreditando em minha capacidade.

*‘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 – Ciclos de Desenvolvimento do Trabalho de Conclusão de Curso	15
Figura 2 – Gráfico de Gantt	15
Figura 3 – Processo de Medição da ISO 15939	17
Figura 4 – Modelo de Informação da ISO 15939	18
Figura 5 – Modelo de Qualidade do Produto da ISO 25023	20

Lista de tabelas

Tabela 1 – Nome dos Intervalos de Frequência	23
Tabela 2 – Intervalos das Métricas para Java e C++	24

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>
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>
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

1	Introdução	13
1.1	Objetivos	14
1.1.1	Objetivos Gerais	14
1.1.2	Objetivos Específicos	14
1.2	Sobre o Trabalho	14
1.2.1	Metodologia de Pesquisa	14
1.2.2	Cronograma	15
1.2.3	Organização do Trabalho	15
2	Métricas	17
2.1	Processo de Medição	17
2.2	Classificação das Métricas	19
2.3	Métricas de Código-Fonte	20
2.3.1	Métrica de Tamanho e Complexidade	21
2.3.2	Métricas de Orientação à Objetos	22
2.3.3	Intervalos da Métricas	23
3	Ferramentas	25
3.1	SonarQube	25
3.1.1	Escolha do SonarQube	25
3.1.2	Limitações Observadas	26
3.2	Pentaho	27
	Referências	28

1 Introdução

As metodologias ágeis vem ganhando cada vez 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 a atividade chave durante um projeto de software (BECK, 1999). Isso também se torna perceptível por algumas práticas do XP ¹ :

1. Padronização do Código: O código é a principal forma de comunicação entre a equipe, logo a padronização de código o torna consistente e fácil para todo o time ler e refatorar.
2. Propriedade Coletiva do Código: Cada programador pode melhorar qualquer parte do código quando existir a oportunidade.
3. Programação em Pares: Todo código é escrito com 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 ou o novo código é descartado.

Logo em metodologias ágeis, é possível inferir a qualidade do trabalho desenvolvido, por meio do código-fonte. Um dos métodos mais utilizados é a análise estática de código, o qual segundo Emanuelsson e Nilsson (2008) significa método automático de coleta de propriedades de tempo de execução do código-fonte sem executá-lo e os resultados da análise podem servir para propósitos de verificação de erros, geração automática de casos de teste, análise de impacto ou ainda para obtenção de métricas de código-fonte.

Embora a definição formal da análise estática consolidada no âmbito acadêmico, vide os trabalhos de (WICHMANN et al., 1995) e (NIELSON; NIELSON; HANKIN, 1999), as ferramentas disponíveis no mercado, que realizam as coletas e avaliação, ainda não suprem totalmente as necessidades de quem precisa avaliar a qualidade total do produto.

Este fato ocorre, pois as ferramentas apresentam os seguintes problemas:

¹ <http://www.extremeprogramming.org/>

- P1 - Ausência de resultados consolidados do produto, pois a maior parte das métricas é extraída de elementos internos menores (Bibliotecas, Pacotes, Classes, Métodos, Funções).
- P2 - Ausência de mecanismos de tratamento, separação e organização e persistência de dados.
- P3 - Ausência de associação entre resultados numéricos e forma de interpretá-los: Ferramentas de análise estática frequentemente mostram seus resultados como valores numéricos isolados para cada métrica (MEIRELLES, 2013).

1.1 Objetivos

Esta seção apresenta os objetivos gerais e específicos deste TCC.

1.1.1 Objetivos Gerais

Sob prisma da avaliação da qualidade de código em ambientes ágeis, neste trabalho, há como objetivo geral a proposição e construção um ambiente de *Dwing*, de modo a melhorar extração e visualização das métricas de código-fonte, suprimindo os pontos fracos enunciados acima das ferramentas de análise estática de código.

Este ambiente inicialmente construído para este trabalho de conclusão de curso, poderá ser expandido em trabalhos futuros para a coleta, transformação e correlação com outras métricas coletadas durante o processo de desenvolvimento de software.

1.1.2 Objetivos Específicos

Os objetivos específicos desse trabalho são:

1. Construir o ambiente de *Dwing* para extração e visualização de métricas de código-fonte.
2. Incorporar indicadores qualitativos para cada métrica de código-fonte.
3. Facilitar o entendimento das métricas de código-fonte.

1.2 Sobre o Trabalho

1.2.1 Metodologia de Pesquisa

A metodologia de pesquisa foi definida em:

Aplicada: pois visa construir conhecimento para aplicações práticas dirigidos á solução de problemas específicos (GIL, 2008).

Metodológica

1.2.2 Cronograma

O Trabalho de Conclusão de Curso foi planejado, como um planejamento de longo prazo, em quatro ciclos de desenvolvimento de vinte e um dias. As figuras 1 e 2, que foram extraídas do OpenProj², apresentam o cronograma geral 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 1 – Ciclos de Desenvolvimento do Trabalho de Conclusão de Curso

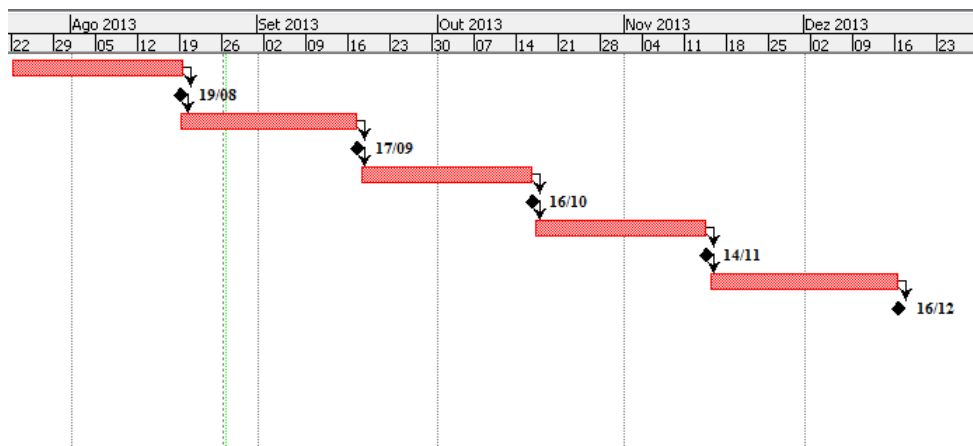


Figura 2 – Gráfico de Gantt

Para cada ciclo de desenvolvimento do Trabalho de Conclusão de Curso, utilizou-se o Quadro *Kanban*

1.2.3 Organização do Trabalho

Para a primeira fase deste Trabalho de Conclusão de Curso, além desta introdução este texto está organizado em capítulos. O Capítulo 2 apresenta o processo de medição

² <http://sourceforge.net/projects/openproj/>

e as métricas de código-fonte. O Capítulo 3 apresenta o SonarQube como a ferramenta escolhida para extração das métricas de código-fonte. O Capítulo 4 apresenta a fundamentação teórica do ambiente de Dwing, de modo a suprir as limitações das ferramentas, e cada um de seus componentes em detalhes. O Capítulo 5 apresenta o ambiente proposto por este trabalho para extração e visualização das métricas de código-fonte. Por fim, o Capítulo 6 apresenta o estado atual da validação do ambiente por meio de estudo de caso, bem como as atividades planejadas até o encerramento do trabalho de conclusão de curso.

2 Métricas

2.1 Processo de Medição

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).

Segundo ISO/IEC 15939 (2002), medição, que é uma ferramenta primordial para gerenciar as atividades do desenvolvimento de software e para avaliar a qualidade dos produtos e a capacidade de processos organizacionais, é um conjunto de operações que visam por meio de um objeto determinar um valor a uma medida ou métrica.¹

O processo de medição da ISO/IEC 15939 (2002), como é mostrado na figura 3, que fora extraído de Gava et al. (2006), consiste de quatro atividades que são sequenciadas em um ciclo iterativo, permitindo feedback e melhoria contínua do processo. Trata-se de uma adaptação do ciclo PDCA (Plan-Do-Check-Act), comumente utilizado como base para a melhoria da qualidade (BARCELLOS, 2010). O processo visa a construção de produtos de informação para cada necessidade seguindo um modelo de informação, que é mostrado na figura 4.

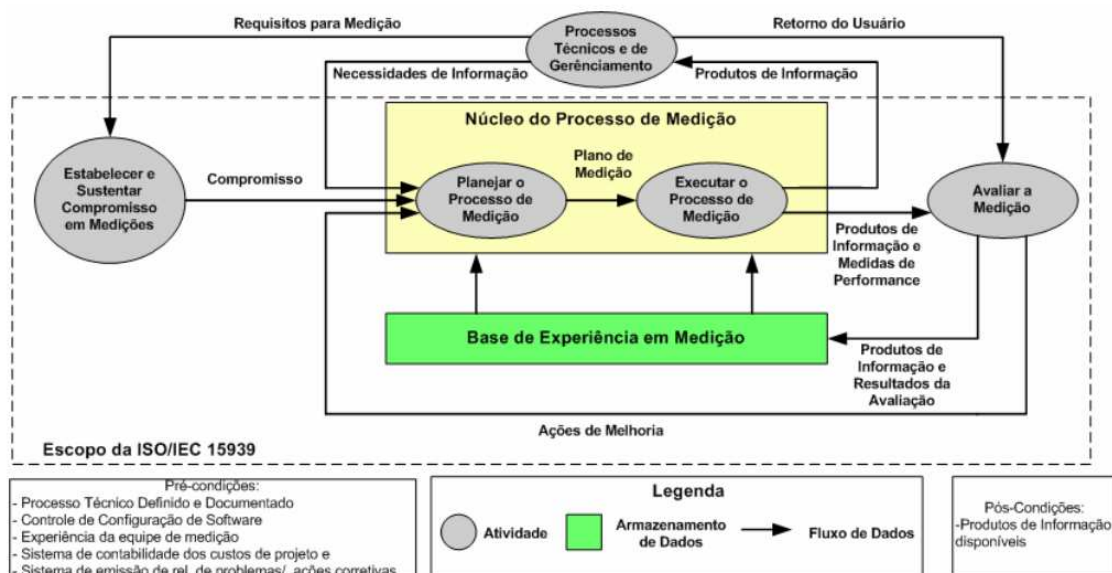


Figura 3 – Processo de Medição da ISO 15939

¹ 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

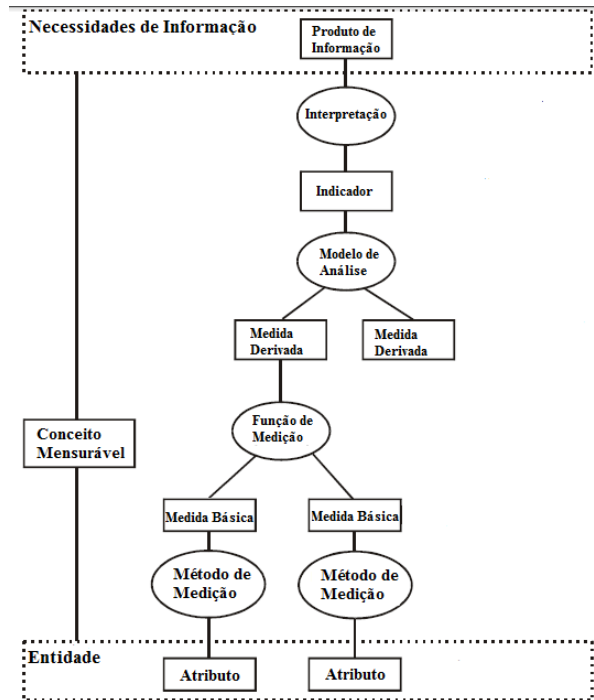


Figura 4 – Modelo de Informação da ISO 15939

A terminologia do modelo de informação, que é mostrado na figura 4 é descrito a seguir:

Necessidade de Informação. São situações as quais requer conhecimento de uma ou mais entidades para gerenciar objetivos, metas, riscos e problemas.

Entidade. Uma entidade é um objeto (como por exemplo, um processo, produto, projeto ou recurso) que é caracterizado por um atributo mensurável;

Atributo. Um atributo é uma propriedade ou característica de uma entidade que pode distingui-la quantitativamente ou qualitativamente utilizando métodos manuais ou automáticos

Conceito Mensurável. Uma relação entre os atributos e as necessidades de informação. Como por exemplo, a qualidade de um produto.

Medida Básica ou Métrica Direta. Uma medida básica é definida em termos do atributo e do método para sua quantificação, ou seja, é a variável a qual se atribui um valor. Cada medida básica é funcionalmente independente de outras medidas básicas e captura apenas um atributo.

Método de Medição. O método de medição é uma sequência de operações, descritas genericamente, usadas qualificar ou quantificar um atributo, ou seja, obter uma medida básica ou derivada com respeito a uma escala. Um método de medição pode ser classificado em:

Subjetivo. Envolve o julgamento humano. Resulta em medidas ou métricas subjetivas.

Objetivo. Baseado em regras numéricas, tais como contagem. Pode ser realizado de forma manual ou automática. Resulta em medidas ou métricas objetivas.

Escala. Uma escala é um conjunto ordenado de valores, contínuos ou discretos, ou uma série de categorias as quais um atributo é mapeado. O método de medição mapeia a magnitude, ou valor absoluto, de um atributo medido em um valor de escala. As escalas podem ser:

Nominal. A medição é categórica. Por exemplo, a classificação dos tipos de defeitos. Nesta escala, só é possível realização de comparações.

Ordinal. A medição é baseada em ordenação. Por exemplo, a classificação de atletas de uma prova de atletismo.

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.

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.

Função de Medição. Trata-se do algoritmo para combinar duas ou mais medidas básicas ou métricas diretas.

Medida Derivada ou Métrica Derivada. Uma medida derivada ou métrica derivada é definida como uma função de medição de duas ou mais medidas básicas ou métricas diretas.

Indicador. É uma medida que fornece uma estimativa ou avaliação de atributos especificados em relação a uma necessidade de informação. Um indicador inclui um ou mais valores de medidas básicas e/ou derivadas.

Interpretação. É a explicação dos indicadores, isto é, a interpretação que cada valor quantitativo do indicador mostra.

Produto de Informação É o resultado esperado do processo de medição, que visa responder as necessidades de informação. O produto de informação pode vir a se materializar em forma de relatórios, gráficos ou planilhas.

2.2 Classificação das Métricas

O modelo proposto pela [ISO/IEC 15939 \(2002\)](#) concentra-se na identificação da necessidade de informação sobre alguma entidade (processo ou projeto). Daí advém a forma mais genérica de classificação, 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* ou *métricas subjetivas*.

Segundo o modelo de qualidade da [ISO/IEC 25023 \(2011\)](#) (Fig. 5), as métricas de produto podem ser subdivididas em três categorias :

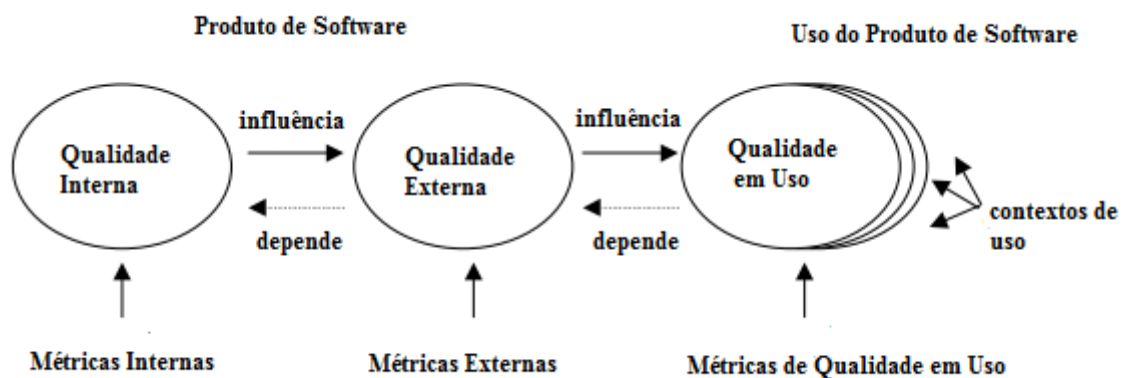


Figura 5 – Modelo de Qualidade do Produto da ISO 25023

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 o qual fora previamente projetado.

2.3 Métricas de Código-Fonte

Segundo a chamada de trabalhos do SCAM 2013², código-fonte é qualquer descrição completamente executável de um sistema de software, desde de linguagem de máquina até representações gráficas executáveis.

As métricas internas de produto ou métricas de código fonte são obtidas por meio da análise das estruturas internas do código sem realizar sua execução (WICHMANN et al., 1995) (NIELSON; NIELSON; HANKIN, 1999) (SOMMERVILLE, 2010). Popularmente, o termo análise estática de código se refere à uma análise automatizada de obtenção de métricas de código fonte (TERRA; BIGONHA, 2008) (EMANUELSSON; NILSSON, 2008).

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:

Tamanho e Complexidade. O tamanho do código-fonte foi uma das 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, por este motivo, a seção 2.3.1 apresenta as métricas de tamanho e complexidade.

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. Para cada paradigma, são necessárias métricas específicas. A seção 2.3.2 apresenta algumas das métricas do paradigma orientado a objetos, pois este ainda é o paradigma mais utilizado no mercado de desenvolvimento de software.

² 13th IEEE International Working Conference on Source Code Analysis and Manipulation

2.3.1 Métrica 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

ACC (*Afferent Connections per Class*) Conexões Aferentes por Classe é número total de classes externas de um pacote que dependem de classes de dentro desse pacote. Quando calculada no nível da classe, essa medida também é conhecida como *Fan-in* da classe, medindo o número de classes das quais a classe é derivada e, assim, valores elevados indicam uso excessivo de herança múltipla (MCCABE; DREYER; WATSON, 1994) (CHIDAMBER; KEMERER, 1994).

RFC (*Response For a Class*) Respostas para uma Classe é número de métodos dentre todos os métodos que podem ser invocados em resposta a uma mensagem enviada por um objeto de uma classe (SHARBLE; COHEN, 1993).

LCOM4 (*Lack of Cohesion in Methods*) Falta de Coesão entre Métodos. Originalmente proposto por Chidamber e Kemerer (1994) como LCOM, contudo essa não teve uma grande aceitabilidade. Após críticas e sugestões a métrica revisada por Hitz e Montazeri (1995), que propôs a LCOM4. Para calcular LCOM4 de um módulo, é necessário construir um gráfico não-orientado em que os nós são os métodos e atributos de uma classe. Para cada método, deve haver uma aresta entre ele e um outro método ou variável que ele usa. O valor da LCOM4 é o número de componentes fracamente conectados nesse gráfico.

NOM (*Number of Methods*) Número de Métodos é usado para medir o tamanho das classes em termos das suas operações implementadas. Essa métrica é usada para ajudar a identificar o potencial de reúso de uma classe. Em geral, as classes com um grande número de métodos são mais difíceis de serem reutilizadas, pois elas são propensas a serem menos coesas (LORENZ; KIDD, 1994).

DIT (*Depth of Inheritance Tree*) Profundidade da Árvore de Herança é o número de superclasses ou classes ancestrais da classe sendo analisada. São contabilizadas apenas as superclasses do sistema, ou seja, as classes de bibliotecas não são contabilizadas. Nos casos onde herança múltipla é permitida, considera-se o maior caminho da classe até uma das raízes da hierarquia. Quanto maior for o valor DIT, maior é o número de atributos e métodos herdados, e portanto maior é a complexidade (SHIH et al., 1997).

NOC (*Number of Children*) Número de Filhos é o número subclasses ou classes filhas que herdam da classe analisada (ROSENBERG; HYATT, 1997). Deve se ter cautela ao modificar classes com muitos filhos, pois uma simples modificação de assinatura de um método, pode criar uma mudança em muitas classes.

2.3.3 Intervalos da Métricas

Em sua tese Meirelles (2013) analisou as métricas do código-fonte de 38 projetos de software livre, em um total de 344.872 classes das aplicações mais bem sucedidas (*Chrome, Firefox, OpenJDK, VLC e entre outros*) e percebeu que há certos valores que são frequentemente encontrados quando se analisa o código-fonte de aplicações que utilizam a mesma linguagem de programação. Meirelles (2013) 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	Intervalo 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 10 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 17]	[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]	[de 0 a 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++

3 Ferramentas

3.1 SonarQube

O SonarQube ¹, 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

3.1.1 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.

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.

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

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.

3.1.2 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.

3.2 Pentaho

Referências

- BARCELLOS, M. P. Medição de software - um importante pilar da melhoria de processos de software. *Engenharia de Software Magazine*, 2010. Citado na página 17.
- BASILI, V. R.; CALDIERA, G.; ROMBACH, H. D. *The Goal Question Metric Approach*. [S.l.]: Encyclopedia of Software Engineering, 1996. Citado na página 17.
- BECK, K. *Extreme Programming Explained*. [S.l.]: Addison Wesley, 1999. Citado na página 13.
- 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 22.
- 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 21.
- FENTON, N. E.; PFLEEGER, S. L. *Software Metrics: A Rigorous and Practical Approach*. 2 edition. ed. [S.l.]: Course Technology, 1998. 656 p. Citado na página 17.
- GAVA, V. L. et al. Modelo do processo de medição de software: pré-condições para sua aplicação. In: *XXVI Encontro Nacional de Engenharia de Produção*. [S.l.: s.n.], 2006. Citado na página 17.
- GIL, A. C. *Métodos e técnicas de pesquisa social*. 6 edition. ed. [S.l.]: Atlas, 2008. Citado na página 15.
- 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 22.
- ISO/IEC 15939. *ISO/IEC 15939: Software Engineering - Software Measurement Process*. [S.l.], 2002. Citado 3 vezes nas páginas 17, 19 e 20.
- 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 na página 20.
- JONES, T. C. *Applied Software Measurement: Assuring Productivity and Quality*. New York: McGraw-Hill, 1991. Citado na página 21.
- 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 21.
- LORENZ, M.; KIDD, J. *Object-Oriented Software Metrics*. [S.l.]: Prentice Hall, 1994. Citado na página 22.

- MCCABE, T. J. A Complexity Measure. *IEEE Transactions Software Engineering*, v. 2, n. 4, p. 308–320, December 1976. Citado na página 22.
- MCCABE, T. J.; DREYER, L. A.; WATSON, A. H. Testing An Object-Oriented Application. *Journal of the Quality Assurance Institute*, v. 8, n. 4, p. 21–27, October 1994. Citado na página 22.
- 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 3 vezes nas páginas 14, 21 e 23.
- 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 20 e 21.
- 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 21.
- 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 22.
- 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 22.
- 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 22.
- SOMMERVILLE, I. *Software Engineering*. 9. ed. Harlow, England: Addison-Wesley, 2010. ISBN 978-0-13-703515-1. Citado na página 21.
- TERRA, R.; BIGONHA, R. S. Ferramentas para análise estática de códigos java. Departamento de Ciência da Computação - UFMG, 2008. Citado na página 21.
- WICHMANN, B. et al. Industrial perspective on static analysis. *Software Engineering Journal*, 1995. Citado 2 vezes nas páginas 13 e 21.