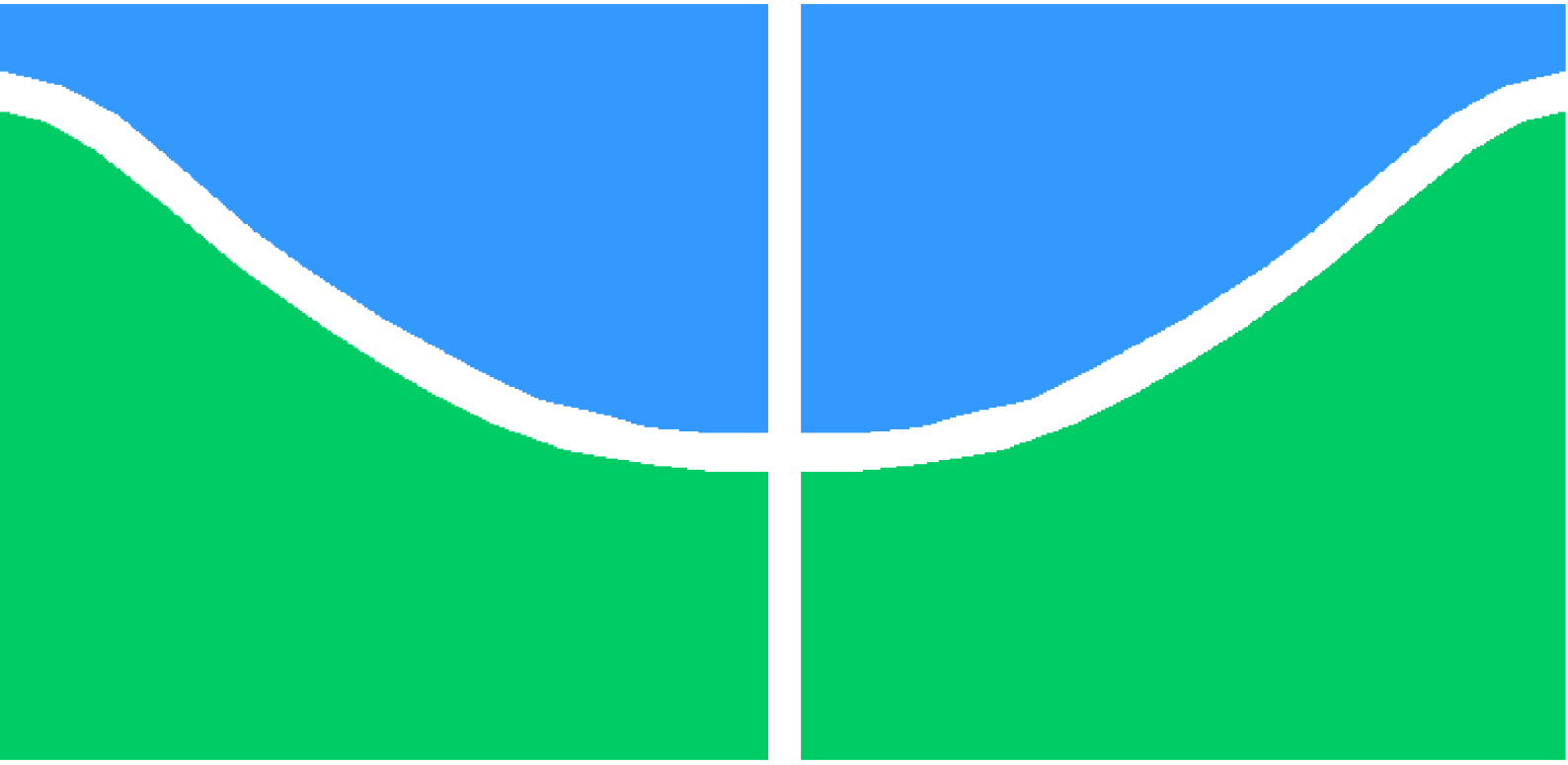

Universidade de Brasília - UnB

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-

35 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 Visualização 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 Neri e Prof. Paulo Meirelles por todo o apoio concedido durante o Trabalho de Conclusão de Curso. Sem qualquer um dos dois, este trabalho não seria possível.

Agradeço aos meus pais, Juno Rego e Andrea Sousa Araújo Baufaker, pelo dom da vida, pela paciência, pela compreensão, pelo apoio incondicional e sobretudo por mostrar que a educação é um dos únicos bens duráveis na vida.

Agradeço à Oracina de Sousa Araújo, minha avó, que sempre me proporcionou conversas muito bem humoradas sobre a forma de ver o mundo, a vida e o ser humano.

Agradeço à Luisa Helena Lemos da Cruz por a compreensão, paciência, dedicação e afeto. A esta mulher cabe toda a minha fonte de dedicação e inspiração para construção daquilo que ainda está por vir.

Agradeço a Tina Lemos e Valdo Cruz pelo apoio e por me receber tão bem quanto um filho é recebido em sua própria casa.

Agradeço a Prof. Eneida Gonzales Valdez por ter me incentivado, com meios de uma verdadeira educadora, a enfrentar os desafios pessoais que a disciplina de Desenho Industrial Assistido por Computador me proporcionou durante as três vezes que a cursei.

Agradeço ao colega de curso, Marcos Ramos, por ter me transmitido conhecimentos acerca de algumas das ferramentas cruciais ao desenvolvimento do trabalho.

Agradeço aos colegas do LAPPIS - Laboratório de Pesquisa Produção e Inovação em Software - por todo conhecimento e horas de reflexão sobre questões cruciais do desenvolvimento de software.

'The only way of discovering the limits of the possible is to venture a little way past them into the impossible."

(Arthur C. Clarke)

Resumo

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

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

Abstract

This is the english abstract.

Key-words: latex. abntex. text editoration.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

ACC	*Afferent Connections per Class*
ACCM	*Average Cyclomatic Complexity per Method*
DIT	*Depth of Inheritance Tree*
DW	*Data Warehouse*
DWing	*Data Warehousing*
GQM	*Goal-Question-Metric*
IEC	*International Electrotechnical Commission*
ISO	*International Organization for Standardization*
LCOM4	*Lack of Cohesion in Methods*
LOC	*Lines of Code*
NOC	*Number of Children*
NOM	*Number of Methods*
PDCA	*Plan-Do-Check-Act*
RFC	*Response For a Class*
SCAM	*IEEE International Working Conference on Source Code Analysis and Manipulation*
XP	*eXtreme Programming*

Sumário

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, Funções) 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 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, consequentemente, 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.

1.2 Sobre o Trabalho

1.2.1 Metodologia de Pesquisa

A metodologia de pesquisa foi definida em **aplicada**, pois visa construir conhecimento, em aplicações práticas, dirigida à solução de problemas específicos (GIL, 2008); **pesquisa-ação** pois visa efetuar transformações nas práticas no mercado de desenvolvimento de software (BROWN; DOWLING, 1998); **quantitativa**, dado que é necessário o uso de técnicas estatísticas para calcular as métricas de código-fonte do software livre do estudo de caso; para, por fim, emitir uma análise **qualitativa** sobre a visualização das métricas de código-fonte em ambientes de DWing.

1.2.2 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 1 e 2, que foram extraídas da ferramenta OpenProj[2], apresentam o cronograma geral e o gráfico de Gantt do presente trabalho.

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". 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).

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*,[3] e foi dividido

[2] Documentação e *download* disponível em `http://sourceforge.net/projects/openproj/`

[3] Disponível em `https://kanbanflow.com`

	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

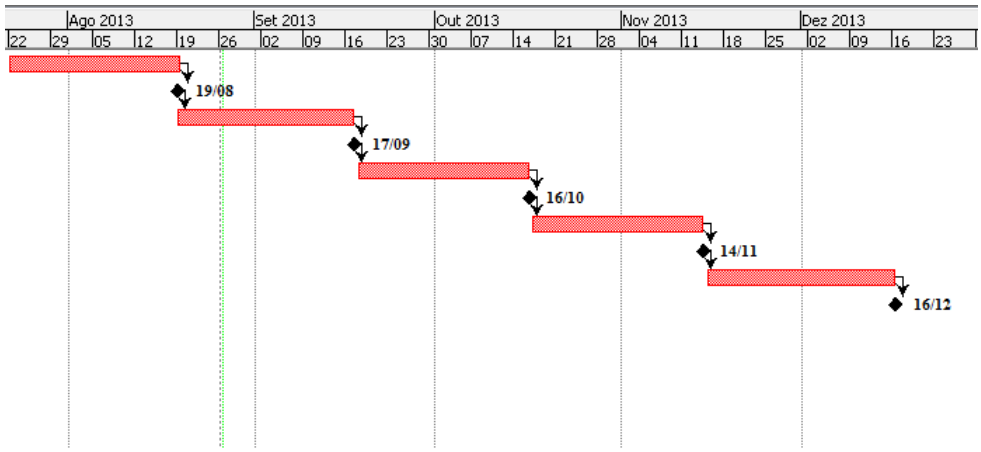

Figura 2 – Gráfico de Gantt do Trabalho de Conclusão de Curso

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 3 mostra o *Kanban* ao final do segundo ciclo de desenvolvimento.

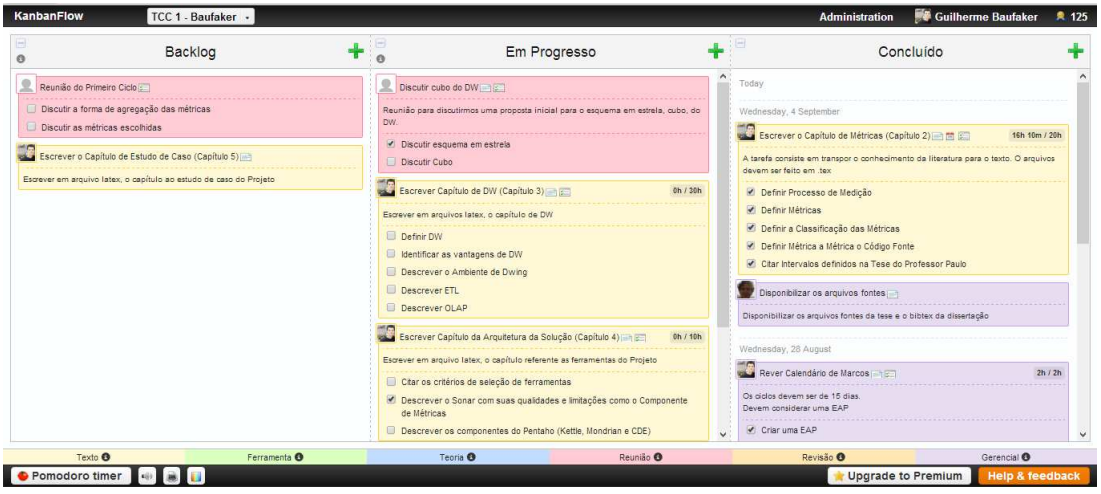

Figura 3 – Quadro *Kanban* do Trabalho de Conclusão de Curso

Todas as tarefas necessárias para realização deste trabalho de conclusão de curso foram listadas no apêndice A.1.

1.2.3 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 o processo de medição, descrito pela ISO 15939, classificação e as métricas de código-fonte, bem como os respectivos intervalos de referência.

O Capítulo 3 apresenta a arquitetura do ambiente de *Dwing*, determinada pelos requisitos de projeto, e composta por: *Data Warehouse(DW)*, processo de ETL *(Extraction-Transformation-Load*, pelo Kettle como componente para realização do processo ETL, pelo SonarQube como fonte de coleta de métricas de código-fonte, pelo esquema estrela com as tabelas fato e dimensão, pelo processo de OLAP, pelo Mondrian como componente para realizá-lo e pela visualização das métricas de código-fonte com o componente CDE para realizá-la.

O Capítulo 4 apresenta um estudo de caso com ambiente de Dwing proposto no capítulo 3.

Por fim, o capítulo 5 apresenta as conclusões retiradas do ambiente.

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 4, 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 5.

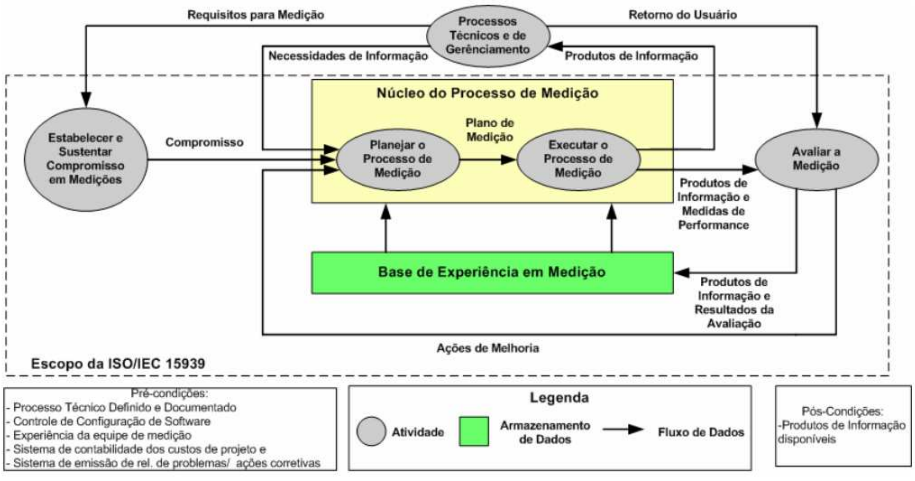

Figura 4 – Processo de Medição da ISO 15939

A terminologia do modelo de informação, que é mostrado na figura 5 é descrito a seguir:

[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

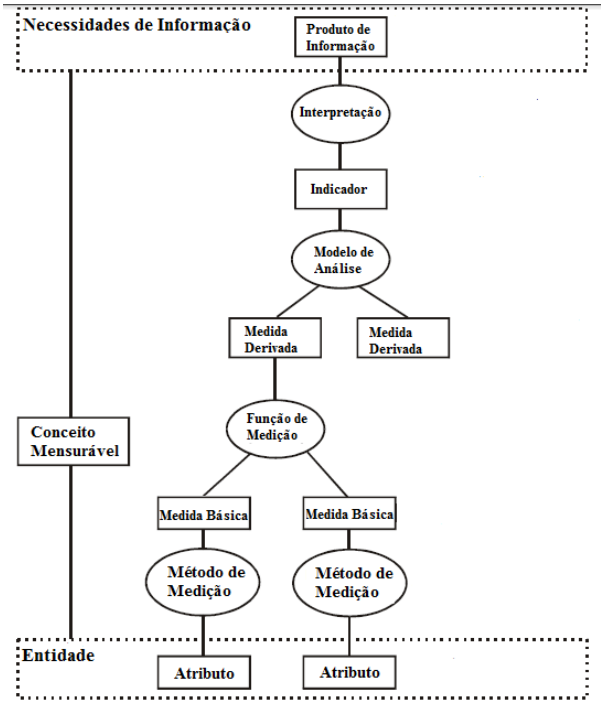

Figura 5 – Modelo de Informação da ISO 15939

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 distinguí-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 1º 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. 6), as métricas de produto podem ser subdivididas em três categorias :

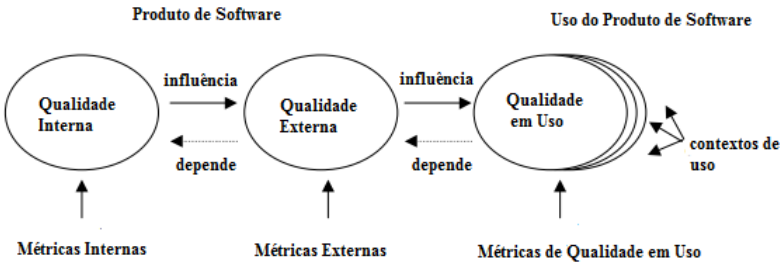

Figura 6 – Modelo de Qualidade do Produto da ISO 25023

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 2013 [2], 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

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

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.

2.3.1 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 do 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 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	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 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++

3 Dwing

3.1 Requisitos de Projeto

3.2 Data Warehouse

3.2.1 Esquema Estrela

3.3 Processo de ETL e Kettle

3.4 Fonte de Coleta das Métricas de Código-Fonte

3.5 Processo de OLAP

4 Solução

4.1 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

4.1.1 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.

4.1.2 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.

4.2 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 18.

BASILI, V. R.; CALDIERA, G.; ROMBACH, H. D. *The Goal Question Metric Approach*. [S.l.]: Encyclopedia of Software Engineering, 1996. Citado na página 18.

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.

BROWN, A.; DOWLING, P. *Doing Research/reading Research: A Mode of Interrogation for Education*. Falmer Press, 1998. (Master classes in education series). ISBN 9780750707282. Disponível em: <http://books.google.com.br-/books?id=3649AAAAIAAJ>. Citado na página 15.

CASTELLANOS, M. et al. ibom: A platform for intelligent business operation management. In: *Proceedings of the 21st International Conference on Data Engineering*. Washington, DC, USA: IEEE Computer Society, 2005. (ICDE '05), p. 1084–1095. ISBN 0-7695-2285-8. Disponível em: <http://dx.doi.org/10.1109/ICDE.2005.73>. Citado na página 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 23.

CHULANI, S. et al. Metrics for managing customer view of software quality. In: *Proceedings of the 9th International Symposium on Software Metrics*. Washington, DC, USA: IEEE Computer Society, 2003. (METRICS '03), p. 189–. ISBN 0-7695-1987-3. Disponível em: <http://dl.acm.org/citation.cfm?id=942804.943748>. Citado na página 14.

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 22.

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 18.

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.

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 18.

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 23.

ISO/IEC 15939. *ISO/IEC 15939: Software Engineering - Software Measurement Process*. [S.l.], 2002. Citado 3 vezes nas páginas 18, 20 e 21.

ISO/IEC 25023. *ISO/IEC 25023: Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - Measurement of system and software product quality*. [S.l.], 2011. Citado na página 21.

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

LEHMAN, M. M. Programs, life cycles, and laws of software evolution. *Proc. IEEE*, v. 68, n. 9, p. 1060–1076, September 1980. Citado na página 22.

LORENZ, M.; KIDD, J. *Object-Oriented Software Metrics*. [S.l.]: Prentice Hall, 1994. Citado na página 23.

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

MCCABE, T. J.; DREYER, L. A.; WATSON, A. H. Testing An Object-Oriented Application. *Journal of the Quality Assurance Institute*, v. 8, n. 4, p. 21–27, October 1994. Citado na página 23.

MEIRELLES, P. R. M. *Monitoramento de métricas de código-fonte em projetos de software livre*. Tese (Doutorado) — Instituto de Matemática e Estátistica – Universidade de São Paulo (IME/USP), 2013. Citado 3 vezes nas páginas 14, 22 e 24.

MILLS, E. E. Metrics in the software engineering curriculum. *Ann. Softw. Eng.*, J. C. Baltzer AG, Science Publishers, Red Bank, NJ, USA, v. 6, n. 1-4, p. 181–200, abr. 1999. ISSN 1022-7091. Citado 2 vezes nas páginas 21 e 22.

MOURA, R. *Kanban: a simplicidade do controle da produção*. [S.l.]: IMAM, 1999. Citado na página 15.

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.

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 23.

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 15.

SHARBLE, R.; COHEN, S. The Object-Oriented Brewery: A Comparison of Two Object-Oriented Development Methods. *Software Engineering Notes*, v. 18, n. 2, p. 60–73, 1993. Citado na página 23.

SHIH, T. et al. Decomposition of Inheritance Hierarchy DAGs for Object-Oriented Software Metrics. In: *Workshop on Engineering of Computer-Based Systems (ECBS 97)*. [S.l.: s.n.], 1997. p. 238. Citado na página 23.

SILVEIRA, P. S.; BECKER, K.; RUIZ, D. D. Spdw+: a seamless approach for capturing quality metrics in software development environments. *Software Quality Control*, Kluwer Academic Publishers, Hingham, MA, USA, v. 18, n. 2, p. 227–268, jun. 2010. ISSN 0963-9314. Disponível em: <http://dx.doi.org/10.1007/s11219-009-9092-9>. Citado na página 14.

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 22.

WICHMANN, B. et al. Industrial perspective on static analysis. *Software Engineering Journal*, 1995. Citado 2 vezes nas páginas 13 e 21.

Apêndices

APÊNDICE A – Tarefas

A.1 Kanban Flow

Neste apêndice são apresentadas as tarefas, na ferramenta Kanban Flow[1] que foram necessárias para execução deste trabalho de conclusão de curso.

[1] Disponível em https://kanbanflow.com

TCC 1 - Baufaker

Backlog (2 tasks)

Reunião do Primeiro Ciclo — Reunião

Responsible: Hilmer Rodrig Time estimate: 0h Time spent: 0h

- [ ] Discutir a forma de agregação das métricas
- [ ] Discutir as métricas escolhidas

Escrever o Capítulo de Estudo de Caso (Capítulo 5) — Texto

Escrever em arquivo latex, o capítulo ao estudo de caso do Projeto

Responsible: Guilherme Bai Time estimate: 0h Time spent: 20h

Em Progresso (5 tasks)

Discutir cubo do DW — Reunião

Reunião para discutirmos uma proposta inicial para o esquema em estrela, cubo, do DW.

Responsible: Hilmer Rodrig Time estimate: 0h Time spent: 0h

- [x] Discutir esquema em estrela
- [ ] Discutir Cubo

Escrever Capítulo de DW (Capítulo 3) — Texto

Escrever em arquivos latex, o capítulo de DW

Responsible: Guilherme Bai Time estimate: 30h Time spent: 0h

- [ ] Definir DW
- [ ] Identificar as vantagens de DW
- [ ] Descrever o Ambiente de Dwing
- [ ] Descrever ETL
- [ ] Descrever OLAP

Escrever Capítulo da Arquitetura da Solução (Capítulo 4)

Texto

Escrever em arquivo latex, o capítulo referente as ferramentas do Projeto

Responsible: Guilherme Bau Time estimate: 10h Time spent: 0h

- [ ] Citar os critérios de seleção de ferramentas
- [x] Descrever o Sonar com suas qualidades e limitações como o Componente de Métricas
- [ ] Descrever os componentes do Pentaho (Kettle, Mondrian e CDE)

Preparar o Kettle para o Estudo de Caso

Ferramenta

Responsible: Guilherme Bau Time estimate: 50h Time spent: 33h 5m

- [x] Extração de Dados Via JSON
- [ ] Persistência no Banco de Dados

Preparar o CDE

Ferramenta

Estudar e preparar o Componente de Visualização do Pentaho

Responsible: Guilherme Bau Time estimate: 0h Time spent: 0h

Labels: Capacitação

Concluído (7 tasks)

Escrever o Capítulo de Métricas (Capítulo 2)

Texto

A tarefa consiste em transpor o conhecimento da literatura para o texto. O arquivos devem ser feito em .tex

Responsible: Guilherme Bau Time estimate: 20h Time spent: 16h 10m
Grouping date: 4 Septembe

~~Due on *2013-09-04T02:00:00Z* for target *Concluído*~~ (Was done by Guilherme Baufaker at 2013-09-04T11:38:43Z)

- [x] Definir Processo de Medição
- [x] Definir Métricas
- [x] Definir a Classificação das Métricas
- [x] Definir Métrica a Métrica o Código Fonte
- [x] Citar Intervalos definidos na Tese do Professor Paulo

Disponibilizar os arquivos fontes

Gerencial

Disponibilizar os arquivos fontes da tese e o bibtex da dissertação

Responsible: Paulo Meirelle Time estimate: 0h Time spent: 0h
Grouping date: 4 Septembe

Rever Calendário de Marcos

Gerencial

Os ciclos devem ser de 15 dias.
Devem considerar uma EAP

Responsible: Guilherme Bai Time estimate: 2h Time spent: 2h
Grouping date: 28 August 2

- [x] Criar uma EAP
- [x] Reorganizar os Ciclos

Reunião com o Marcos Ramos

Reunião

Reunião com o Marcos Ramos às 10h no LAPPIS com intuito de aprender o CDE

Responsible: Guilherme Bai Time estimate: 2h Time spent: 2h 14m
Grouping date: 23 August 2

~~Due on *2013-08-23T15:00:00Z* for target *Concluído*~~ (Was done by Guilherme Baufaker at 2013-08-23T15:14:15Z)

Criar Cronograma de Marcos

Gerencial

Criar o Cronograma de Marcos do TCC.

Data Prevista da Defesa: 16/12.

25% do Semestre: 17/09/2013
50% do Semestre: 16/10/2013
75% do Semestre: 14/11/2013
100% do Semestre (Data da Defesa): 16/12/2013

Responsible: Guilherme Bai Time estimate: 1h 30m Time spent: 2h
Grouping date: 23 August 2

Criar Estrutura de Diretórios

Ferramenta

Criar estrutura de diretórios no repositório Git. Separando os arquivos texto dos arquivos do Pentaho.

Responsible: Guilherme Bai Time estimate: 20m Time spent: 25m
Grouping date: 23 August 2

Achar Ferramenta de Kanban

Ferramenta

Procurar alguma ferramenta que tivesse suporte ao quadro kanban.

Responsible: Guilherme Ba Time estimate: 30m Time spent: 15m
Grouping date: 22 August 2

~~Due on *2013-08-23T02:00:00Z* for target *Concluído*~~ (Was done by Guilherme Baufaker at 2013-08-23T01:59:57Z)

- [x] Criar Conta na Ferramenta
- [x] Convidar Hilmer e Paulo