



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

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

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

Brasília, DF
2013



Guilherme Baufaker Rêgo

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

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

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: Prof. Msc. Hilmer Rodrigues Neri

Coorientador: Prof. Dr. Paulo Roberto Miranda Meirelles

Brasília, DF

2013

Guilherme Baufaker Rêgo

Extração e Visualização de Métricas de Código-Fonte em um ambiente de *Data Warehousing*/ Guilherme Baufaker Rêgo. – Brasília, DF, 2013-
60 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Msc. Hilmer Rodrigues Neri

Coorientador: Prof. Dr. Paulo Roberto Miranda Meirelles

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2013.

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

CDU 02:141:005.6

Guilherme Baufaker Rêgo

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

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

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

Prof. Msc. Hilmer Rodrigues Neri
Orientador

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

**Prof. Dr. Rodrigo Bonifácio de
Almeida**
Convidado 1

**Msc. Débora Reinaldo Arnoud Lima
Formiga**
Convidado 2

Brasília, DF
2013

Este trabalho é dedicado a minha avó paterna, Isaura da Silva, e minha bisavó materna, Enedina Gaspar de Sousa Araújo. Estas foram os exemplos de meus exemplos.

Agradecimentos

Agradeço aos meu orientador Prof. Hilmer Neri por ter sido um professor que estendeu o meu aprendizado sobre o desenvolvimento de software a muito além da sala de aula. A ele sou grato, desde a oportunidade concedida no projeto Desafio Positivo, que foi um divisor de águas em minha formação profissional quanto a oportunidade de conhecer desde *data warehouse*, metodologias ágeis e outras áreas importantes da engenharia de software.

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

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

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

Agradeço à Oracina de Sousa Araújo, minha avó materna, que sempre me proporcionou conversas muito bem humoradas sobre a forma de ver o mundo, a vida e o ser humano. Além da minha maior parte da criação, devo a ela desde todo o suporte fornecido em casa até a preocupação sobre a quantidade de horas de sono durante a noite.

Agradeço à Luisa Helena Lemos da Cruz por ser tão compreensiva, paciente, amorosa e dedicada para comigo, sobretudo nos dias que atencederam a entrega desde trabalho. A ela devo todas as transformações positivas de minha vida desde de setembro 2012 até então. Sem sombras de dúvidas, ela é minha fonte de dedicação, inspiração para construir o futuro.

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

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

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

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

Resumo

As ferramentas de análise estática isoladas não proveem bons mecanismos de visualização de dados para transmitir a informação da qualidade do principal produto do desenvolvimento de software, o código-fonte. Após uma revisão bibliográfica da literatura, identificou-se que ambientes de *data warehousing* (DWing) são boas soluções em consultas e visualização de dados. Partindo desta premissa, este trabalho construiu um ambiente de DWing para visualização e extração de métricas de código-fonte, que são indicadores objetivos da qualidade deste. Por meio da realização de um estudo de caso de avaliação das métricas de código-fonte de um software livre no ambiente de DWing, verificou-se que ambientes de *DWing* proveem melhores mecanismos de visualização e consulta de dados.

Palavras-chaves: Métricas de Código-Fonte. *Data Warehousing*. *Data Warehouse*

Abstract

Statical Analysis Tools isolated do not provide good data visualization on main product of software development: the source code. After a bibliografic review on literature, it has been identified that Data Warehousing (DWing) is a good enviroment to provide solution on data visualization. Taking it as a premise, this work has built a DWing enviroment in order to visualize and extract source code metrics, which are objective indicators of quality on source code. Using this enviroment built, it has been performed a study case of source code metrics evaluation of an open source software. By the study case, it was possible to verify that data warehousing provide better resources of visualization and consulting data.

Palavras-chaves: *Source Code Metrics. Data Warehousing. Data Warehouse*

Lista de ilustrações

Figura 1 – Quadro <i>Kanban</i> do Trabalho de Conclusão de Curso	16
Figura 2 – Modelo de Informação da ISO 15939 extraído de Gomes (2011)	17
Figura 3 – Modelo de Qualidade do Produto da ISO 25023 adaptado de ISO/IEC 25023 (2011)	19
Figura 4 – Arquitetura de um ambiente de <i>Data Warehousing</i> extraído de Rocha (2000)	24
Figura 5 – Exemplo de Esquema Estrela extraído de Times (2012)	26
Figura 6 – Exemplo de Cubo Multidimensional de dados extraído de Times (2012)	27
Figura 7 – Exemplo de Dashboard extraído de Minardi (2013)	32
Figura 8 – Outro exemplo de Dashboard extraído de Minardi (2013)	32
Figura 9 – Arquitetura do Ambiente de DWing para Métricas de Código-Fonte . .	33
Figura 10 – Comparação entre JSON, YAML e XML extraído de Fonseca e Simoes (2007)	37
Figura 11 – Modelo Multidimensional do <i>Data Warehouse</i>	41
Figura 12 – Interface do Kettle	42
Figura 13 – Interface do Pentaho BI Platform	44
Figura 14 – Visualização de dados do Apache Maven em formato de tabela no Saiku Analytics	47
Figura 15 – Visualização de dados do Apache Maven em formato de gráfico de barras no Saiku Analytics	48
Figura 16 – Calendário de Marcos da próxima etapa deste Trabalho de Conclusão de Curso	49
Figura 17 – <i>Job</i> no Kettle	56
Figura 18 – Primeira <i>Trasformation</i> do ETL	56
Figura 19 – Segunda <i>Transformation</i> do ETL	57
Figura 20 – Obtendo os Dados do JSON	58
Figura 21 – Resultado do Processamento do <i>JSON input</i>	59
Figura 22 – Resultado do Processamento Estatístico das Métricas de Código-Fonte	59
Figura 23 – Terceira <i>Transformation</i> do ETL	60
Figura 24 – Quarta <i>Transformation</i> do ETL	60

Lista de tabelas

Tabela 1 – Modelo de Informação para metodologias ágeis com base na ISO/IEC 15939 (2002)	18
Tabela 2 – Nome dos Intervalos de Frequência	22
Tabela 3 – Intervalos das Métricas para Java e C++	23
Tabela 4 – Diferenças entre OLAP e OLTP extraído de Times (2012), Rocha (2000) e Neri (2002)	28
Tabela 5 – Exemplo do Total de Vendas de uma Rede de Lojas no mês de Novembro	29
Tabela 6 – Exemplo do Total de Vendas de uma rede de lojas no mês de novembro com a dimensão Produto	29
Tabela 7 – Exemplo do Total de vendas da loja norte no mês de novembro	29
Tabela 8 – Exemplo de Vendas por produto de uma rede de lojas nos meses de novembro e dezembro	30
Tabela 9 – Exemplo de Vendas do Produto A na rede de lojas	30
Tabela 10 – Exemplo de Vendas por loja para cada um dos produtos nos meses de novembro e dezembro	31
Tabela 11 – Critérios Gerais de seleção de ferramentas	34
Tabela 12 – Critérios Específicos para Ferramenta de Análise Estática de Código-Fonte	34
Tabela 13 – Características do SonarQube e do Analizo	35
Tabela 14 – Análise do SonarQube e do Analizo quanto aos critérios gerais e quanto aos critérios específicos de ferramentas de análise estática	36
Tabela 15 – Entidades do Negócio	40
Tabela 16 – Características do Kettle e avaliação quanto aos critérios gerais de seleção de ferramentas	43
Tabela 17 – Características do Pentaho BI Platform e avaliação quanto aos critérios gerais de seleção de ferramentas	44
Tabela 18 – Características do Saiku Analytics e avaliação quanto aos critérios gerais de seleção de ferramentas	45
Tabela 19 – Métricas de Código Fonte do Apache Maven em 07/09/2013	54
Tabela 20 – Métricas de Código Fonte do Apache Maven em 13/10/2013	54
Tabela 21 – Métricas de Código Fonte do Apache Maven em 20/10/2013	54
Tabela 22 – Métricas de Código Fonte do Apache Maven em 27/10/2013	55
Tabela 23 – Métricas de Código Fonte do Apache Maven em 03/11/2013	55
Tabela 24 – Métricas de Código Fonte do Apache Maven em 10/11/2013	55

Lista de abreviaturas e siglas

ACC	<i>Afferent Connections per Class</i>
ACCM	<i>Average Cyclomatic Complexity per Method</i>
DIT	<i>Depth of Inheritance Tree</i>
DW	<i>Data Warehouse</i>
DWing	<i>Data Warehousing</i>
ETL	<i>Extraction-Transformation-Load</i>
GQM	<i>Goal-Question-Metric</i>
IEC	<i>International Electrotechnical Commission</i>
ISO	<i>International Organization for Standardization</i>
LCOM4	<i>Lack of Cohesion in Methods</i>
LOC	<i>Lines of Code</i>
NOC	<i>Number of Children</i>
NOM	<i>Number of Methods</i>
OLAP	<i>On-Line Analytical Processing</i>
OLTP	<i>Online Transaction Processing</i>
RFC	<i>Response For a Class</i>
SCAM	<i>IEEE International Working Conference on Source Code Analysis and Manipulation</i>
SGBD	<i>Sistema de Gerenciamento de Bancos de Dados</i>
XP	<i>eXtreme Programming</i>

Sumário

1	Introdução	14
1.1	Objetivos	15
1.1.1	Objetivo Geral	15
1.1.2	Objetivos Específicos	15
1.2	Metodologia de Pesquisa	16
1.3	Organização do Trabalho	16
2	Métricas de Software	17
2.1	Processo de Medição de Software	17
2.2	Classificação das Métricas de Software	18
2.3	Métricas de Código-Fonte	19
2.3.1	Métrica de Tamanho e Complexidade	20
2.3.2	Métricas de Orientação à Objetos	20
2.3.3	Intervalos das Métricas	21
3	Data Warehousing (DWing)	24
3.1	<i>Extraction-Transformation-Load</i> (ETL)	25
3.2	<i>Data Warehouse</i>	25
3.2.1	Metodologia do Projeto do <i>Data Warehouse</i>	27
3.3	<i>On-Line Analytical Processing</i> (OLAP)	27
3.4	Visualização de Dados	31
4	Implementação do Ambiente de DWing	33
4.1	Arquitetura da Implementação	33
4.2	Ferramenta de Análise Estática de Código-Fonte	34
4.2.1	Estudo de caso de Acompanhamento das Métricas de Código-Fonte de um Software Livre em ambiente de DWing	37
4.3	Projeto do <i>Data Warehouse</i>	38
4.3.1	Entidades do Negócio	39
4.3.2	Projeto Lógico	41
4.4	Ferramentas de DWing	42
4.4.1	Implementação da Extração, Transformação e Carga dos Dados	42
4.4.2	Implementação das Consultas OLAP e Visualização de Dados	43
5	Considerações Finais	47
5.1	Resultados Parciais	47

5.2 Trabalhos Futuros	49
Referências	50
APÊNDICE A Métricas de Código-Fonte do Apache Maven	54
APÊNDICE B Descrição Simplicada do Processo de ETL no Kettle	56

1 Introdução

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

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

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

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

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

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

Dado este contexto, é crucial que dados relacionados às métricas sejam coletados e compartilhados entre projetos e pessoas em uma visão organizacional unificada, para que determinada organização ou time possa compreender o processo de medição e monitoramento de projetos de software e, conseqüentemente, se tornar mais hábil e eficiente em realizar atividades técnicas relacionadas ao processo de desenvolvimento de software (CHULANI et al., 2003).

Vários trabalhos têm mostrado que ambientes de *Data Warehousing* (DWing) são boas soluções para atender à visão organizacional unificada de métricas de software proposta por Chulani et al. (2003). Vide os trabalhos de Palza, Fuhrman e Abran (2003), Ruiz et al. (2005), Castellanos et al. (2005), Becker et al. (2006), Folleco et al. (2007), Silveira, Becker e Ruiz (2010).

Tendo os trabalhos, enunciados anteriormente, como o referencial teórico para ambientes de *Data Warehousing*, adotou-se como questão de pesquisa deste trabalho:

Q1 - A visualização das métricas de código-fonte em ambientes de DWing confere maior visibilidade da qualidade geral do código-fonte de um determinado projeto?

Com o intuito de verificar a questão de pesquisa, adotou-se como hipótese de pesquisa que **visualização e a extração de métricas de código-fonte em ambientes de DWing possibilitam as equipes maior poder de análise sobre a qualidade geral do código fonte quando comparada com as análises providas por ferramentas de análise estática de código-fonte isoladamente.**

1.1 Objetivos

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

1.1.1 Objetivo Geral

Sob o prisma da questão de pesquisa, o objetivo geral deste trabalho é construir a visualização de métricas de código-fonte em um ambiente de *Data Warehousing* (DWing) a fim de se validãr a hipótese de pesquisa.

1.1.2 Objetivos Específicos

Os objetivos específicos deste trabalho são:

OE1 - Construir o ambiente de DWing com ferramentas de software livre.

OE2 - Analisar a qualidade de um software livre a fim de instanciar a visualização das métricas de código-fonte em ambientes de DWing.

1.2 Metodologia de Pesquisa

A metodologia de pesquisa deste trabalho foi construída em etapas. Para a primeira etapa, realizou-se pesquisa bibliográfica para a construção do referencial teórico.

Na segunda etapa, realizou-se um estudo comparativo entre ferramentas livres para a construção do ambiente de *Data Warehousing*.

Após o levantamento das ferramentas, construiu-se o ambiente *Data Warehousing*, em pequenos ciclos de produção. Para o acompanhamento das atividades de implementação, utilizou-se o Quadro *Kanban*, que foi inventado pela Toyota com objetivo de controlar o fluxo da produção, mostrado na Figura 1.

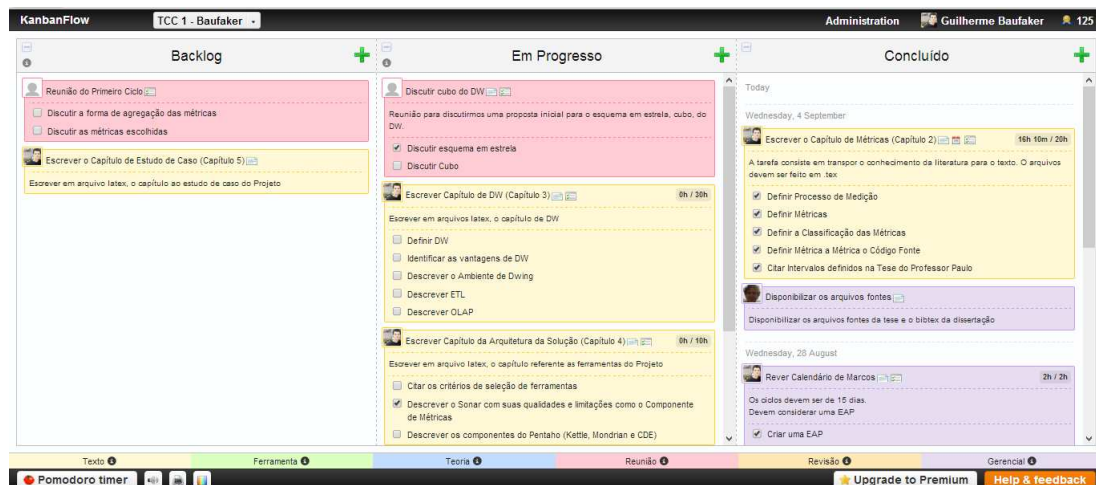


Figura 1 – Quadro *Kanban* do Trabalho de Conclusão de Curso

Após a construção do ambiente de *Data Warehousing*, foi realizado um estudo de caso de coleta de métricas de código-fonte de um software livre com intuito de verificar a hipótese de pesquisa.

1.3 Organização do Trabalho

Para a primeira fase deste Trabalho de Conclusão de Curso, além desta introdução, o texto foi organizado em capítulos. O Capítulo 2 apresenta as métricas de software bem

como o processo de medição, descrito pela ISO 15939, e as métricas de código-fonte. O Capítulo 3 apresenta conceitualmente o ambiente de *data warehousing* (DWing). O Capítulo 4 apresenta a implementação do ambiente de DWing construído neste trabalho. Por fim, o capítulo 5 apresenta as considerações finais com uso do ambiente de DWing na extração e visualização de métricas de código-fonte.

2 Métricas de Software

2.1 Processo de Medição de Software

Segundo [Fenton e Pfleeger \(1998\)](#), medição é o mapeamento de relações empíricas em relações formais. Isto é, quantificação em símbolos com objetivo de caracterizar uma entidade por meio de seus atributos. Contrapondo-se com uma definição operacional, a [ISO/IEC 15939 \(2002\)](#) define medição como conjunto de operações que visam por meio de um objeto determinar um valor a uma medida ou métrica ¹. Alguns modelos de referência, como [CMMI \(2010\)](#), e até a própria [ISO/IEC 15939 \(2002\)](#) definem medição como uma ferramenta primordial para gerenciar as atividades do desenvolvimento de software e para avaliar a qualidade dos produtos e a capacidade de processos organizacionais.

A [ISO/IEC 15939 \(2002\)](#) define um processo de medição com base em um modelo de informação, que é mostrado na Figura 2, a fim de obter produtos de informação para cada necessidade de informação. Para isto, cada necessidade de informação, que é uma situação que requer conhecimento com intuito de gerenciar objetivos, metas, riscos e problemas, é mapeada em uma construção mensurável que tem em sua origem um conceito mensurável, como por exemplo, tamanho, qualidade e custo a fim de mapear um atributo (característica que permite distinguir qualitativamente e/ou quantitativamente) uma entidade que pode ser um processo, projeto ou produto de software. Por fim cada construção mensurável é mapeada em um ou mais produtos de informação que levam uma ou mais métricas ou medidas, que podem ser classificadas sob critérios apresentados na seção 2.2.

¹ A definição formal da [ISO/IEC 15939 \(2002\)](#) não utiliza o termo métrica, sendo que este é utilizado por abordagens mais antigas como GQM em [Basili, Caldiera e Rombach \(1996\)](#), que é uma outra abordagem para medição, contudo compreende-se que o termo medida tem valor semântico equivalente ao de métrica no contexto do deste trabalho

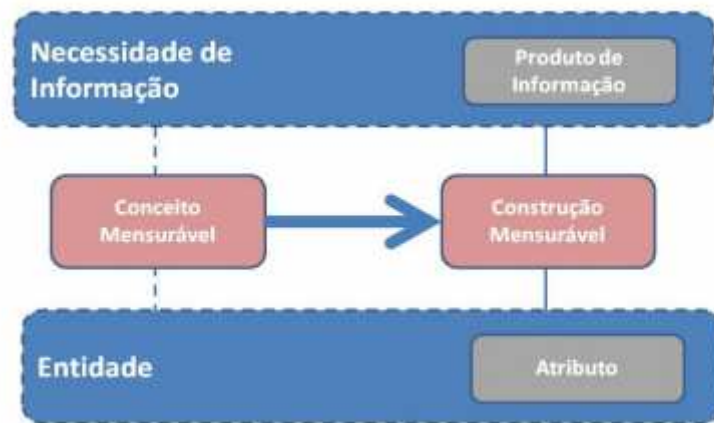


Figura 2 – Modelo de Informação da ISO 15939 extraído de [Gomes \(2011\)](#)

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

Entidade	Código Fonte
Conceito Mensurável	Qualidade
Construção Mensurável	Qualidade do Código-Fonte
Atributos a ser medidos	Classes, Métodos e Pacotes
Produto de Informação	Indicadores de Qualidade do Código-Fonte

Tabela 1 – Modelo de Informação para metodologias ágeis com base na [ISO/IEC 15939 \(2002\)](#)

2.2 Classificação das Métricas de Software

As métricas de software possuem uma escala de medição, que é um conjunto ordenado de valores, contínuos ou discretos, ou uma série de categorias nas 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 unidades. Por exemplo, o aumento 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. 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 ao 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 3, as métricas de produto podem ser subdivididas em três categorias:

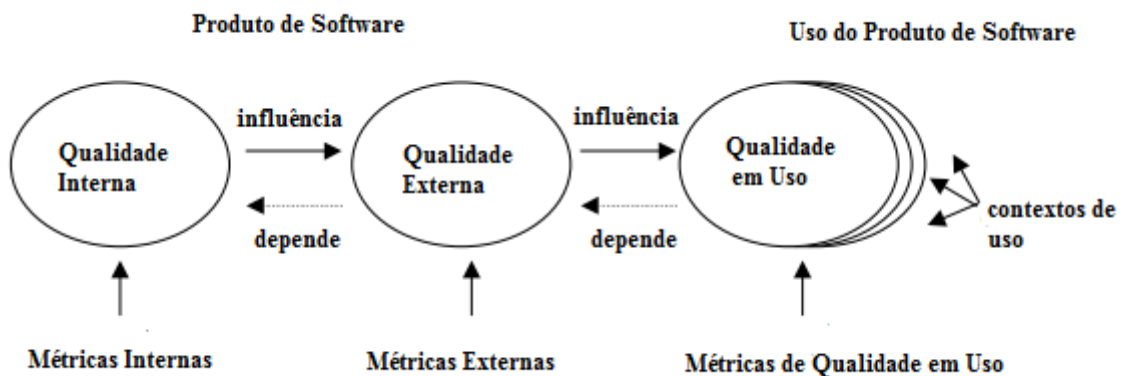


Figura 3 – Modelo de Qualidade do Produto da ISO 25023 adaptado de ISO/IEC 25023 (2011)

Métricas Internas. São métricas que aferem a qualidade interna do software por meio da avaliação de estruturas internas que compõem o software em estágio de desenvolvimento. São conhecidas como métricas de código-fonte.

Métricas Externas. São métricas que capturam o comportamento do software. Exemplos de atributos da qualidade externa: correção, usabilidade, eficiência e robustez. Qualidade externa mede o comportamento do software. Estas só podem ser aferidas

por atividades de teste do desenvolvimento do software em condições similares as que serão encontradas em ambientes de implantação.

Métricas de Qualidade em Uso. São métricas que aferem se o software atende as necessidades do cliente com eficiência, produtividade, segurança e satisfação em contextos específicos de uso. Estas só podem ser coletadas em ambientes reais, isto é, o ambiente de implantação.

2.3 Métricas de Código-Fonte

Segundo a *International Working Conference on Source Code Analysis and Manipulation* (SCAM), o código-fonte é qualquer especificação executável de um sistema de software. Por conseguinte, se inclui desde código de máquina até linguagens de alto nível ou representações gráficas executáveis. (HARMAN, 2010).

Portanto, as métricas de código-fonte, quando obtidas por meio da análise estática de código-fonte, advém de especificações executáveis do software, isto é, trechos de código escritos em alguma linguagem de programação. Isto confere às métricas de código-fonte a classificação de métricas objetivas e características como validade, simplicidade, objetividade, fácil obtenção e robustez (MILLS, 1999). Meirelles (2013) realizou um levantamento sistemático na literatura das métricas de código-fonte que foram categorizadas, nas 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 era impresso. A segunda lei de Lehman (1980) enuncia que a complexidade aumenta à medida que o software é evoluído, a menos que seja um trabalho de manutenção. Logo, é perceptível que as métricas de complexidade estão diretamente ligadas as métricas de tamanho, sendo que a modificação em uma provavelmente impactará na outra. A seguir são apresentadas as principais métricas de tamanho e complexidade:

LOC (*Lines of Code*) Número de Linhas de Código foi uma das primeiras métricas utilizadas para medir o tamanho de um software. São contadas apenas as linhas executáveis, ou seja, são excluídas linhas em branco e comentários. Para efetuar comparações entre sistemas usando LOC, é necessário que ambos tenham sido feitos na mesma linguagem de programação e que o estilo esteja normalizado (JONES, 1991).

ACCM (*Average Cyclomatic Complexity per Method*) Média da Complexidade Ciclométrica 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 ciclométrica 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. Já o paradigma da orientação à objetos visa abstrair as unidades computacionais em Classes, que representam em grande parte do desenvolvimento, unidades reais do negócio. Estas unidades geram 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 o 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 não teve uma grande aceitabilidade. Após críticas e sugestões a métrica foi 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 de 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 das Métricas

No trabalho de Meirelles (2013), foram analisadas métricas do código-fonte de 38 projetos de software livre, em um total de 344.872 classes das aplicações mais utilizadas de software livre, tais como, **Chrome, Firefox, OpenJDK, VLC** e entre outros. Após uma análise estatística com o 75º percentil dos valores obtidos para cada métrica de código-fonte, com exceção do LOC que foi considerado o 50º ou simplesmente a mediana, Meirelles (2013) percebeu que há certos valores que são frequentemente encontrados quando se analisa o código-fonte de aplicações de mesma linguagem de programação. De forma a agregar os valores comuns, Meirelles (2013) classificou as métricas de código-fonte nos seguintes intervalos: *muito frequente, frequente, pouco frequente e não frequente*.

Visando simplificar o entendimento das métricas de código-fonte, renomeou-se os intervalos tal como a Tabela 2. Posteriormente, apresenta-se a Tabela 3, decorrente do estudo de Meirelles (2013), com os intervalos encontrados para C++ e Java.

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

Tabela 2 – Nome dos Intervalos de Frequência

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

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

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

3 *Data Warehousing* (DWing)

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

Data Warehousing (DWing) é uma coleção de tecnologias de suporte à decisão disposta a capacitar os responsáveis por tomar decisões a fazê-las de forma mais rápida (CHAUDHURI; DAYAL, 1997 apud ROCHA, 2000). Em outras palavras, trata-se de um processo para montar e gerenciar dados vindos de várias fontes, com o objetivo de prover uma visão analítica de parte ou do todo do negócio (GARDNER, 1998). Desta forma, é possível em um ambiente de *data warehousing* que as métricas de código-fonte sejam coletadas de fontes diversas em uma periodicidade definida, de forma automatizada, não intrusiva ao trabalho da equipe de desenvolvimento e que estas possam mostrar a qualidade total do código-fonte produzido pela equipe durante um determinado período de tempo (dias, meses, anos).

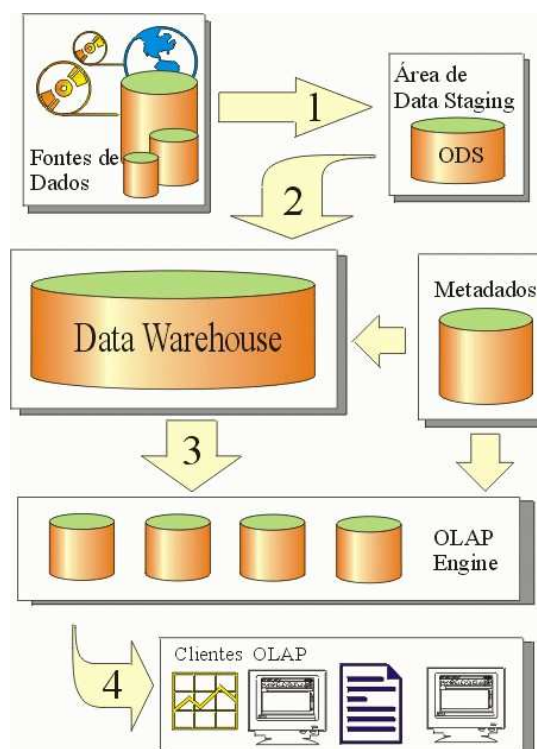


Figura 4 – Arquitetura de um ambiente de *Data Warehousing* extraído de Rocha (2000)

A Figura 4 descreve uma arquitetura geral de um ambiente de DWing, de tal forma que,

- i) As setas 1 e 2 representam o processo de *Extraction-Transformation-Load*;
- ii) A seta 3 representa as consultas *On-Line Analytical Processing (OLAP)*;
- iii) por fim a seta 4 representa a visualização dos dados;

Cada um dos componentes da Figura 4 é descrito nas seções subsequentes.

3.1 *Extraction-Transformation-Load (ETL)*

As etapas de extração, transformação, carga e atualização do *data warehouse* formam o back-end e caracterizam o processo chamado *Extraction- Transform-Load (ETL)*. Esse processo pode ser dividido em três etapas distintas que somadas podem consumir até 85% de todo o esforço em um DWing (KIMBALL; ROSS, 2002).

- Extração: No ambiente de *data warehousing*, os dados, que provêm de fontes distintas, tais como planilhas, bases relacionais em diferentes tipos de banco de dados (MySQL, Oracle, PostgreSQL e etc) ou mesmo de web services, são inicialmente extraídos de fontes externas de dados para um ambiente de *staging* que Kimball e Ross (2002) considera com uma área de armazenamento intermediária entre fontes e o *data warehouse*. Normalmente, é de natureza temporária e o seu conteúdo é apagado após a carga dos dados no *data Warehouse*.
- Transformação: Após os dados serem carregados na área de *staging*, os dados passam por processos de transformações diversas. Estas podem envolver desde uma simples transformação de ponto para vírgula até a realização de cálculos, como por exemplo, cálculos estatísticos.
- Carga: Após as devidas transformações dos dados, os dados são carregados, em formato pré-definido pelo projeto do *data warehouse*, em definitivo no afim de serem utilizados pelas consultas OLAP.

3.2 *Data Warehouse*

Data Warehouse (DW) é um conjunto de dados integrados, consolidados, históricos, segmentados por assunto, não-voláteis, variáveis em relação ao tempo, e de apoio às decisões gerenciais (INMON, 1992), ou seja, trata-se de um repositório central e consolidado que se soma ao conjunto de tecnologias que compõem um ambiente maior, que é o DWing (KIMBALL; ROSS, 2002).

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

trou que a modelagem relacional com a utilização das técnicas de normalização, que visam a eliminação da redundância de dados, não é eficiente quando se realiza consultas mais complexas que fazem uso frequente da operação JOIN entre várias tabelas, pois oneram recursos hardware com grandes quantidades de acesso físico a dados. (KIMBALL; ROSS, 2002)

Dado esse cenário, Kimball e Ross (2002) propôs que o *data warehouse* deve ser projetado de acordo com as técnicas de modelagem dimensional, que visam exibir os dados em níveis adequados de detalhes e otimizar consultas complexas (TIMES, 2012). No modelo dimensional, são aceitos que as tabelas possuam redundância e esparcidade de dados e estas podem ser classificadas em tabelas fatos e tabelas dimensões. Estas contêm dados textuais, que pode conter vários atributos descritivos que expressam relações hierarquizáveis do negócio. Já uma tabela fato é uma tabela primária no modelo dimensional onde os valores numéricos ou medidas do negócio são armazenados (KIMBALL; ROSS, 2002).

Quando se juntam fatos e dimensões desnormalizadas, obtém-se o chamado esquema estrela, tal como se mostra na Figura 5. Quando em um modelo dimensional, se faz necessário uso da normalização, o modelo passa então a ser chamado por *modelo snowflake*, cujo ganho de espaço é menor que 1% do total necessário para armazenar o esquema do *data warehouse* (TIMES, 2012 apud KIMBALL; ROSS, 2002). Em ambos os casos, quando se relaciona três dimensões, obtém-se os cubos de dados (KIMBALL; ROSS, 2002), tal como se mostra na figura 6.

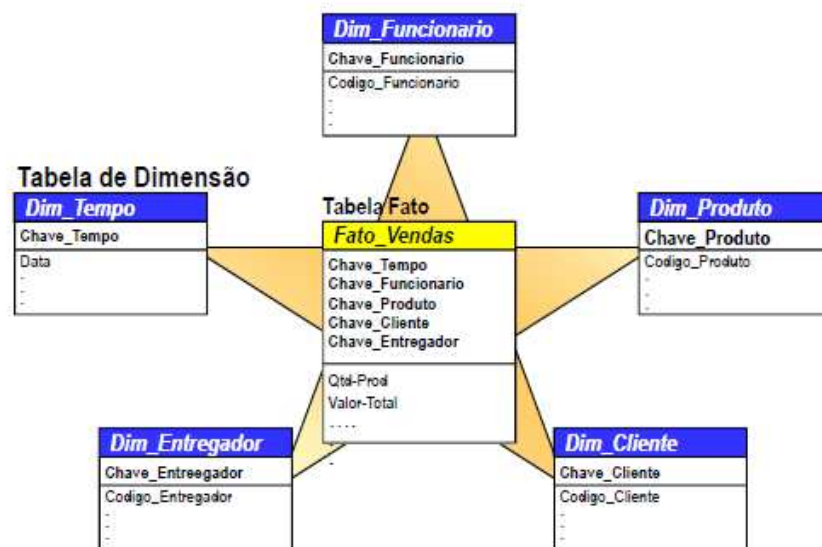


Figura 5 – Exemplo de Esquema Estrela extraído de Times (2012)

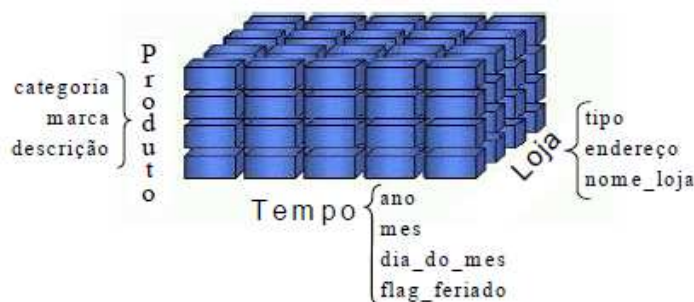


Figura 6 – Exemplo de Cubo Multidimensional de dados extraído de [Times \(2012\)](#)

No esquema da Figura 5, percebe-se que uma tabela fato expressa um relacionamento muitos para muitos com as tabelas dimensões, mostrando assim que a navegabilidade dos dados quantitativos e qualitativos é mais intuitiva quando comparada com o modelo relacional normalizado ([KIMBALL; ROSS, 2002](#)). Além disso, verifica-se que a tabela fato possui uma dimensão temporal associada, isto é, há fatos que ocorrem diariamente, como por exemplo, a venda de produtos em um supermercado. Contudo, é possível que as vendas sejam vistas por visões mensais, trimestrais, semestrais ou anuais. Logo, a granularidade dos fatos deve ser considerada na hora de projetar um *data warehouse*. Além disto, deve-se ainda considerar as características do fato, pois quando os registros de uma tabela fato podem ser somados a qualquer dimensão, é dito que o fato é aditivo. Quando é possível apenas somar em relação a algumas dimensões, é dito que o fato é semiaditivo. Já quando o fato é usado apenas para registro e não pode ser somado em relação a nenhuma dimensão, é dito que o fato é não aditivo ([INMON, 1992](#)).

3.2.1 Metodologia do Projeto do *Data Warehouse*

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

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

3.3 *On-Line Analytical Processing* (OLAP)

O termo OLAP, inicialmente proposto por [Codd, Codd e Salley \(1993\)](#), é utilizado para caracterizar as operações de consulta e análise em um *data warehouse* projetado sobre um modelo dimensional ([KIMBALL; ROSS, 2002](#)). Isto permite consultas mais

flexíveis quando comparadas com as consultas *Online Transaction Processing* (OTLP) que são executadas em bancos de dados relacionais normalizados, visando a eliminação da redundância de dados.

As principais diferenças das operações *On-Line Analytical Processing* (OLAP) para as operações *Online Transaction Processing* (OTLP) são apresentados na Tabela 4.

OLAP	OLTP
Modelagem Dimensional (Tabelas Fato e Dimensão)	Modelagem Relacional com a utilização das formas normais (3N, 4N, 5N)
Dados armazenados em nível transacional e agregado	Dados em nível em nível transacional
Visa o diminuir o uso do JOIN	Faz uso constante de Join
Análise de Dados	Atualização de dados
Estrutura de tipicamente estática	Estrutura tipicamente dinâmica
Proveem informações atuais e do passado	Geralmente sem suporte a estado temporal dos dados

Tabela 4 – Diferenças entre OLAP e OLTP extraído de [Times \(2012\)](#), [Rocha \(2000\)](#) e [Neri \(2002\)](#)

Segundo [Neri \(2002\)](#), a consolidação é uma das mais importantes operações OLAP. Ela envolve a agregação de dados sobre uma ou mais hierarquias de dimensões. A generalização de uma consulta de consolidação pode ser representada formalmente através de:

Select $P, F_1(m_1), \dots, F_p(m_p)$
From $C(D_1(A_{11}), \dots, D_n(A_{n+1}))$
Where $\phi(D_1)$ **and** ... **and** $\phi(D_n)$
Group by G

onde P representa os atributos a serem selecionados das dimensões. $F_i(m_i)$ para $(1 \leq i \leq p)$ representando uma função de agregação. A cláusula **From** $C(D_1(A_{11}), \dots, D_n(A_{n+1}))$ indica que a fonte de dados está indexada por suas tabelas dimensões, sendo que cada uma destas é referenciada como $D_i \dots D_n$ onde D_i contém K_i atributos de $D_i(A_{i1}), \dots, D_i(A_{ik_i})$ que descrevem a dimensão. A cláusula **Where** $\phi(D_i)$ é o predicado $(D_i(A_{ij}) = v_{ij})$, onde $v_{ij} \in \text{dom}(D_i(A_{ij}))$ onde $(1 \leq i \leq n)$ e $(1 \leq j \leq K_i)$. A cláusula **Group by** G $\subset D_i(A_{ij})$ tal que $(1 \leq i \leq n)$ e $(1 \leq j \leq K_i)$.

As operações OLAP tem como objetivo prover visualização dos dados sob diferentes perspectivas gerenciais e comportar todas as atividades de análise. Estas podem

ser feitas de maneira *ad hoc*, por meio das ferramentas de suporte a operações OLAP. Contudo, há algumas que são documentadas pela literatura e são classificadas em dois grupos: Análise Prospectiva e Análise Seletiva (CHAUDHURI; DAYAL, 1997 apud ROCHA, 2000).

A análise prospectiva consiste em realizar a análise a partir de um conjunto inicial de dados para chegar a dados mais detalhados ou menos detalhados (INMON, 1992). Já a análise seletiva tem como objetivo trazer à evidência para os dados (ROCHA, 2000). Entre as operações de análise prospectiva estão:

- *Drill-Down*: Descer no nível de detalhes dos dados de uma dimensão. isto é, adicionar cabeçalhos de linha de tabelas de dimensão (KIMBALL; ROSS, 2002).
- *Roll-Up*: contrário de Drill-Down, trata-se caminhar para a visão de dados mais agregados (KIMBALL; ROSS, 2002 apud ROCHA, 2000).

Considerando o exemplo do total de vendas no mês de novembro em uma rede de lojas, que agregam as Lojas Sul, Norte e Oeste, tal como se mostra a Tabela 5, a operação Drill-Down pode ser exemplificada, quando se adiciona a dimensão Produto na Tabela 5, isto é, aumentando o nível de detalhes, tendo então como resultado a Tabela 6. Já a operação de Roll-Up é o contrário, isto é, diminuir o nível de detalhe partindo da Tabela 6 para Tabela 5.

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

Tabela 5 – Exemplo do Total de Vendas de uma Rede de Lojas no mês de Novembro

Mês	Loja	Produto			
		Produto A	Produto B	Produto C	Produto D
Novembro	Loja Sul	10	70	50	70
Novembro	Loja Norte	100	60	50	90
Novembro	Loja Oeste	25	78	67	60

Tabela 6 – Exemplo do Total de Vendas de uma rede de lojas no mês de novembro com a dimensão Produto

- *Drill-Across*: significa caminhar a partir de uma dimensão para outra dimensão, combinando-as para mudar o enfoque da análise (ROCHA, 2000). O Drill Across pode ser aplicado à Tabela 5, obtendo assim a Tabela 7.

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

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

Entre as operações de análise seletiva estão:

- *Slice and Dice*: Em português, significa cortar e fatiar. Esta operação seleciona pedaços transversais do modelo dimensional e em seguida aplica critérios de seleção sobre este pedaço. (ROCHA, 2000). Ou seja, trata-se de uma operação semelhante a cláusula WHERE do SQL (TIMES, 2012). A operação pode ser aplicada na Tabela 8, obtendo assim a Tabela 9.

Produto	Loja	Outubro	Novembro	Dezembro
Produto A	Loja Sul	50	10	20
	Loja Norte	60	100	24
	Loja Oeste	70	25	53
Produto B	Loja Sul	32	70	20
	Loja Norte	42	60	43
	Loja Oeste	56	78	56
Produto C	Loja Sul	34	50	23
	Loja Norte	45	50	74
	Loja Oeste	83	67	65
Produto D	Loja Sul	56	70	35
	Loja Norte	12	90	34
	Loja Oeste	64	60	23

Tabela 8 – Exemplo de Vendas por produto de uma rede de lojas nos meses de novembro e dezembro

Produto	Loja	Outubro	Novembro
Produto A	Loja Sul	50	10
	Loja Norte	60	100
	Loja Oeste	70	25

Tabela 9 – Exemplo de Vendas do Produto A na rede de lojas

- *Pivoting*: Trata-se de uma operação de rotação de 90° em um cubo multidimensional, isto é, muda-se a orientação das tabelas dimensionais a fim de restringir a visualização das dimensões em uma tabela. (ROCHA, 2000). A operação de Pivoting pode ser exemplificada ao partir da Tabela 8 para Tabela 10.

Loja	Produto	Outubro	Novembro	Dezembro
Loja Sul	Produto A	50	10	20
	Produto B	32	70	20
	Produto C	34	50	23
	Produto D	56	70	35
Loja Norte	Produto A	60	100	24
	Produto B	42	60	43
	Produto C	45	50	74
	Produto D	12	90	34
Loja Oeste	Produto A	70	25	53
	Produto B	56	78	56
	Produto C	83	67	65
	Produto D	64	60	23

Tabela 10 – Exemplo de Vendas por loja para cada um dos produtos nos meses de novembro e dezembro

3.4 Visualização de Dados

Dados transmitem importantes informações, logo cabe a quem deseja comunicá-los, escolher a forma mais efetiva de fazê-lo (MINARDI, 2013). Segundo Minardi (2013), tabelas e gráficos são as formas mais comuns de transmitir as informações quantitativas, sendo que i) tabelas são utilizadas para consulta de valores individuais que podem ser comparados envolvendo, em certos casos, mais de uma unidade de medida; ii) gráficos são indicados para exibição de informação quantitativa nos quais os valores indicam pontos de interesse e estes podem ser comparados por sua similaridades e dessimilaridades.

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

Uma das técnicas de visualização, que visam transmitir os dados efetivamente e que tem se destacado é o uso dos *dashboards*, que são representações visuais das informações mais importantes e necessárias, que podem ser monitoradas simultaneamente, para atingir um ou mais objetivos de negócio (MINARDI, 2013). Minardi (2013) também enuncia que cada componente do *dashboard* deve ser pequeno, conciso, claro e intuitivo de modo que o todo deve caber inteiramente em uma tela de computador, tal como se mostra na Figura 7 e na Figura 8.

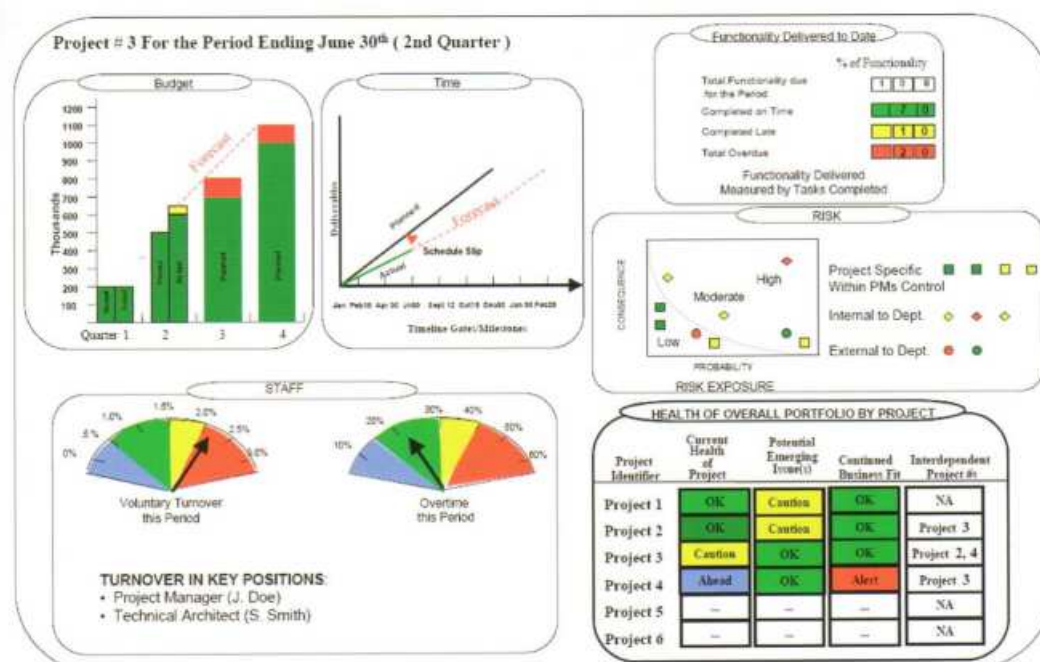


Figura 7 – Exemplo de Dashboard extraído de Minardi (2013)

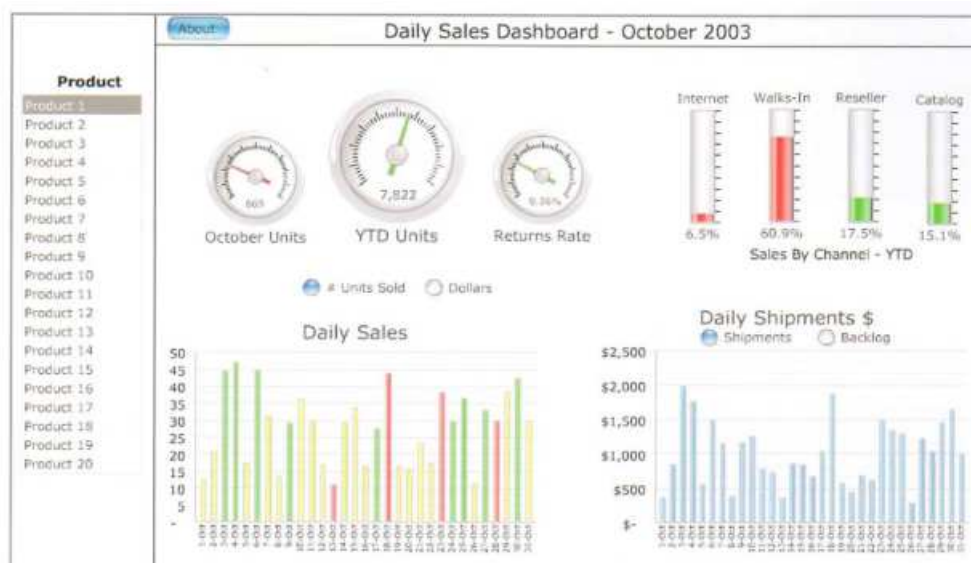


Figura 8 – Outro exemplo de Dashboard extraído de Minardi (2013)

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

4 Implementação do Ambiente de DWing

4.1 Arquitetura da Implementação

Para a implementação do ambiente do DWing para métricas de código-fonte, foi definida a arquitetura tal como se mostra Figura 9.

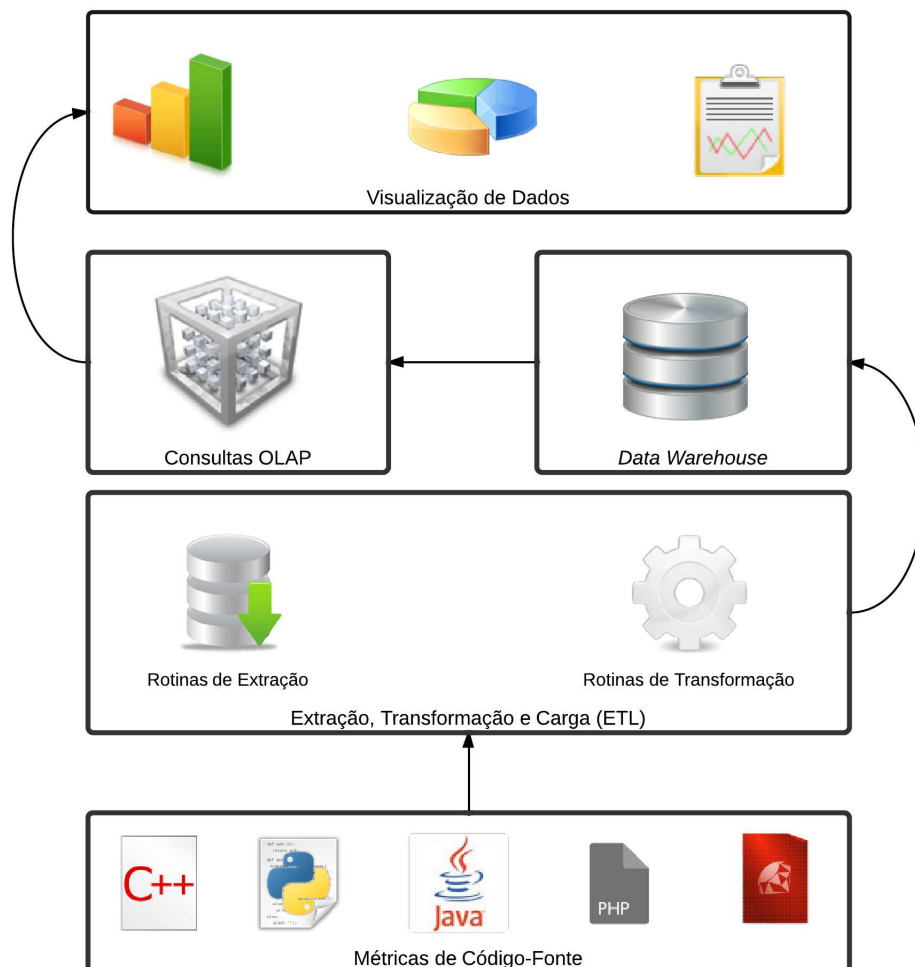


Figura 9 – Arquitetura do Ambiente de DWing para Métricas de Código-Fonte

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

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

Tabela 11 – Critérios Gerais de seleção de ferramentas

4.2 Ferramenta de Análise Estática de Código-Fonte

Além dos critérios gerais estabelecidos para escolha da ferramenta de análise estática de código-fonte, que é a fonte externa de coleta dos dados, estabeleceram-se os critérios específicos para seleção de ferramentas de análise estática de código fonte (CAE) apresentados na Tabela 12.

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

Tabela 12 – Critérios Específicos para Ferramenta de Análise Estática de Código-Fonte

Após a realização de uma busca por ferramentas de análise estática de código-fonte, foram pre-selecionados o SonarQube ¹ e Analizo ² cujas principais características de ambas são apresentadas na Tabela 13.

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

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



Característica		
Linguagens com Suporte	C, C++, Java	C, C++, Java, PHP, Scala, Python, Delphi, Pascal, Flex, ActionScript, Javascript, Groovy ³
Licença	GNU GPL3	GNU LGPL3
Métricas de Código-Fonte fornecidas	9 métricas em âmbito de Projeto e 16 métricas em âmbito de Classe (MEIRELLES, 2013)	12 métricas em âmbito de Classe, 8 métricas em âmbito de projeto.
Formato de Saída das Métricas	YAML	JSON, XML
Plataforma	Windows, Linux, Mac OS X e Servidores de Aplicação Java	Debian e Ubuntu
Integração com outras ferramentas	Mezuro, Kalibro	Jenkins, Hudson, Mantis, JIRA, Crowd e entre outros
Números do Repositório Oficial no GitHub em 14/11/2013	<i>Commits:</i> 639	<i>Commits:</i> 7532
	<i>Branches:</i> 1	<i>Branches:</i> 16
	<i>Contribuidores:</i> 7	<i>Contribuidores:</i> 18
	<i>Releases:</i> 27	<i>Releases:</i> 36
Número de Casos Abertos no <i>Issue Tracker</i> em 14/11/2013	26	552
Sistema de Controle de Versões	Git	Git
Idioma com Suporte	Inglês	Inglês, Português, Japonês, Italiano, Chinês, Francês, Grego e Espanhol
Idioma da Documentação	Inglês	Inglês
Última Versão Estável em 14/11/2013	1.17.0	4.0
Data de Lançamento da Última Versão Estável	31 de Janeiro de 2013	4 de Novembro de 2013

Tabela 13 – Características do SonarQube e do Analizo

Tendo as características gerais de cada ferramenta levantadas, foram comparadas (SonarQube e Analizo) quanto aos critérios gerais e aos critérios específicos para ferramentas de análise estática, tal como se mostra na Tabela 14.

³ O SonarQube oferece suporte comercial a outras linguagens, contudo foram listadas apenas que tem

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

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

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

Devido aos critérios específicos de seleção de ferramentas de análise estática, percebe-se que o Analizo tem uma plataforma específica, sendo que o SonarQube é multi-plataforma. Ainda que o critério não seja atendido pelo Analizo, há o objetivo de construir um ambiente de *Data Warehousing* utilizando ferramentas livres, logo poder-se-ia escolher também um sistema operacional livre tal como o Ubuntu ou Debian.

Dado o cenário apresentado, a escolha foi ponderada então pelo formato de saída do arquivo. [Fonseca e Simoes \(2007\)](#) realizou um estudo experimental entre XML, JSON (Formatos de Saída do SonarQube) e YAML (Formato de Saída do Analizo), que é mostrado na Figura 10.

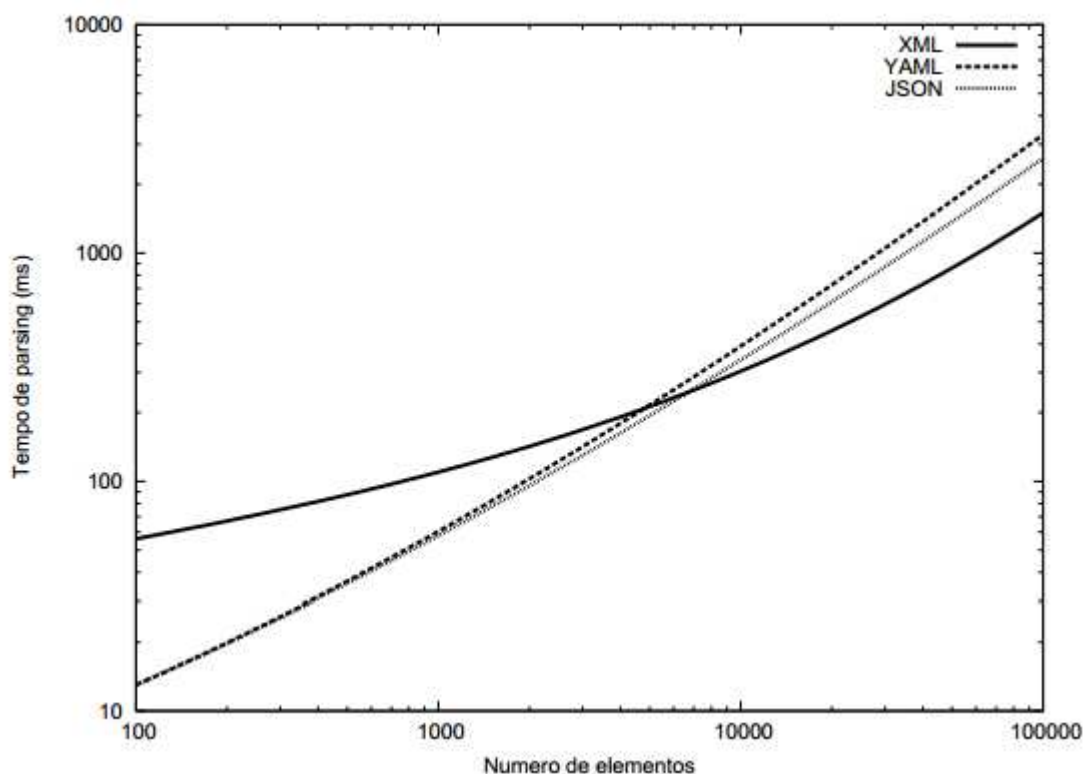


Figura 10 – Comparação entre JSON, YAML e XML extraído de [Fonseca e Simoes \(2007\)](#)

Após a análise da Figura 10, percebe-se que o YAML perde em desempenho para JSON e XML quando o número de elementos aumenta gradativamente. Em projetos de software, é natural que estes cresçam cada vez mais, tal como especifica a segunda lei de [Lehman \(1980\)](#). Logo, o formato JSON ou XML é mais adequado, fazendo com que a ferramenta SonarQube seja mais indicada para o contexto deste trabalho.

Entre JSON e XML, [Fonseca e Simoes \(2007\)](#) conclui que o transporte de dados é mais rápido utilizando JSON, pois se permite transportar um objeto serializado arbitrariamente complexo, e transformá-lo num objeto JavaScript sem grandes problemas, enquanto que no XML se deve construir um *parser* para interpretar os dados. Por este motivo, foi escolhido para extração das métricas de código-fonte neste trabalho o formato JSON provido pela ferramenta SonarQube.

4.2.1 Estudo de caso de Acompanhamento das Métricas de Código-Fonte de um Software Livre em ambiente de DWing

Com o objetivo de validar a implementação do ambiente de DWing para métricas de código-fonte, escolheu-se nesta etapa do trabalho de conclusão de curso acompanhar a qualidade de um software livre por meio de um estudo caso.

Dado que os intervalos constituídos da análise de [Meirelles \(2013\)](#), apresentados

na seção 2.3.3, são para as linguagens de programação C++ e Java, há a restrição quanto ao acompanhamento de projetos feitos nessas linguagens.

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

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

4.3 Projeto do *Data Warehouse*

O *Data Warehouse* como elemento central do ambiente de *Data Warehousing* deve ser o primeiro a ser projetado (KIMBALL; ROSS, 2002). Isso ocorre pois o DW deve ser dirigido ao negócio. Logo a modificação do DW impacta principalmente na carga dos dados, na etapa de extração, transformação e carga, requerendo modificações conforme o DW venha a mudar.

Seguindo a metodologia proposta por Kimball e Ross (2002), apresentada na seção 3.2.1, entende-se que o processo de negócio a ser avaliado é o monitoramento da qualidade do código-fonte expresso por meio das métricas de código-fonte. Este negócio possui algumas demandas, que foram mapeadas, como i) **Comparação das métricas de código-fonte entre projetos de mesma linguagem de programação ao longo de um período de x meses**; ii) **Acompanhamento dos indicadores de qualidade de um determinado projeto ao longo do tempo**. Embora Kimball e Ross (2002) enuncie que todo o negócio deva ser mapeado dentro do desenvolvimento de um ambiente de *Data Warehousing*, compreende-se, por analogia ao desenvolvimento de software, que o levantamento total de requisitos para implementação aumenta exponencialmente o custo da mudança (BECK, 1999b). Por este motivo, adicionalmente a metodologia do Kimball, utilizou-se o princípio ágil de pequenos ciclos de entrega de software. Logo, as demandas do negócio ainda não foram totalmente mapeadas e estas assim o serão até a próxima etapa do trabalho.

Kimball e Ross (2002) enuncia que o segundo passo após a indentificação do processo de negócio é a identificação da periodicidade dos dados coletados pelo mesmo. No

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

processo de negócio de métricas de código-fonte, a periodicidade de coleta dos dados é variável, isto é, depende de projeto a projeto. Visando atender a maior parte dos casos possíveis, identificou-se a menor granularidade para agregação: o dia. Isto ocorre, pois as ferramentas de integração contínua, que são muito utilizadas em ambientes ágeis ([BECK, 1999a](#)), permitem a estabilização de um pacote de software por dia, logo a menor análise do código-fonte pode ser realizada sobre esse pacote estável diariamente. Contudo, há projetos que liberam versões estáveis com mais tempo, portanto se deve permitir agregações maiores tais como, mês, trimestre, semestre e ano. Para esta etapa do trabalho de conclusão de curso, optou-se pela implementação da agregação diária.

Seguindo os passos subsequentes da metodologia proposta por [Kimball e Ross \(2002\)](#), é realizada a identificação das entidades do negócio para por fim realizar modelagem dimensional do *Data Warehouse*.

4.3.1 Entidades do Negócio

Partindo da descrição das demandas que já foram mapeadas, identificou-se entidades do negócio tal como mostra a Tabela [15](#).

Entidade	Atributos	Descrição do Atributo	Aditividade
Projeto	identificador	identificador único do Projeto	não aditivo
	nome	identificador textual do Projeto	não aditivo
Métrica	identificador	identificador único da Métrica	não aditivo
	nome	identificador textual da Métrica	não aditivo
	descrição	descrição textual da métrica	não aditivo
	valor	trata-se da medida, isto é, o valor obtido para cada métrica	aditivo em relação ao tempo
Intervalo	identificador	identificador único do intervalo	não aditivo
	máximo	valor discreto máximo encontrado por	não aditivo
	mínimo	valor discreto mínimo encontrado por	não aditivo
	índice de qualidade	Índice de qualidade para um determinado intervalo, podendo ser Excelente, Bom, Regular ou Ruim conforme especificado na Tabela 3	não aditivo
	linguagem de programação	A linguagem de programação é um atributo do intervalo, pois para cada linguagem de programação, há intervalos diferenciados. No escopo da análise, de Meirelles (2013) há apenas C++ e Java	não aditivo
Tempo	identificador	identificador único do intervalo	não aditivo
	dia	valor discreto do dia de um mês (varia de 1 a 31)	não aditivo
	mês	valor discreto do mês em um ano (varia de 1 a 12)	não aditivo
	ano	valor discreto do dia de um ano (expresso em 4 dígitos)	não aditivo

Tabela 15 – Entidades do Negócio

4.3.2 Projeto Lógico

A partir das entidades do negócio, foram identificados as dimensões e o fato por meio da modelagem dimensional, tal como mostra a Figura 11, utilizando a ferramenta MySQL Workbench ⁵ que é um software de código aberto com licença GPL 2.0.

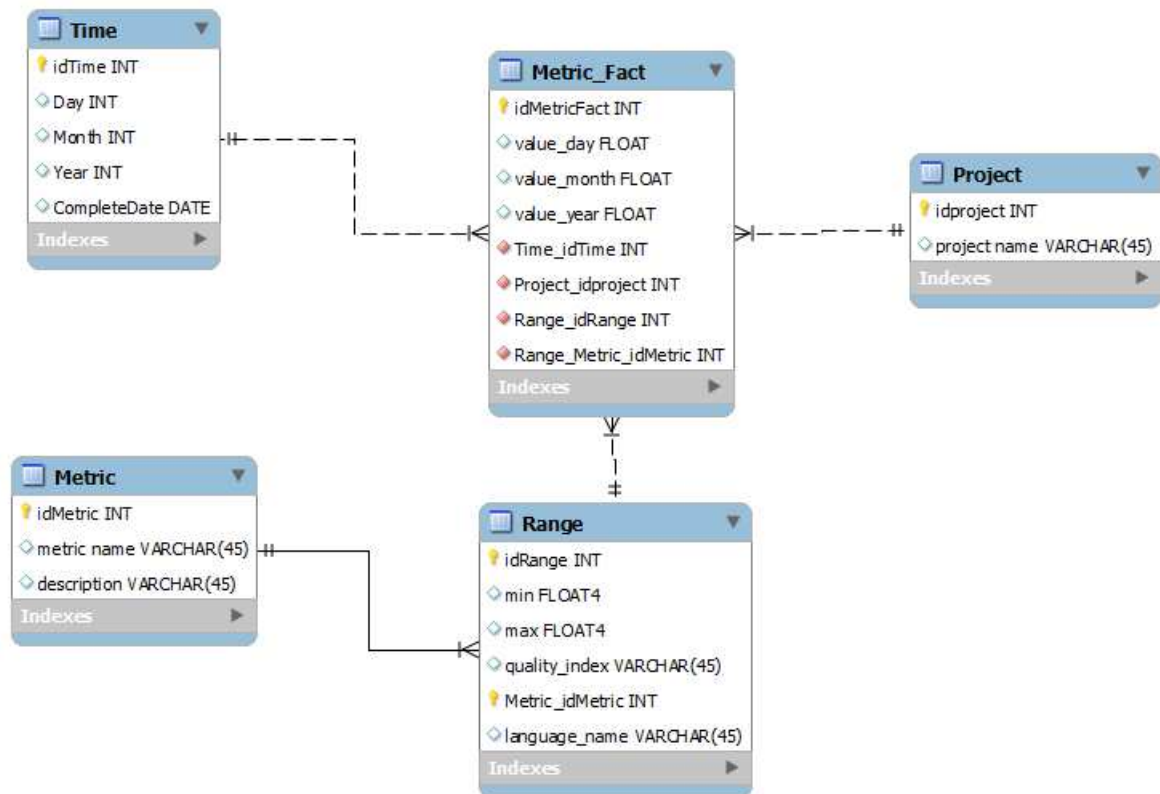


Figura 11 – Modelo Multidimensional do *Data Warehouse*

A dimensão Tempo (representada na tabela Time) foi identificada a partir da periodicidade da coleta. A dimensão Projeto (Project) foi identificada ao se analisar a entidade projeto do negócio. As dimensões Métrica (Tabela Metric) e Intervalo (Range) também foram identificadas a partir das entidades de negócio. Meirelles (2013) enuncia que: para cada métrica específica há um intervalo específico para uma determinada linguagem de programação. Em vista da restrição, utilizou-se a normalização resultando em um *snowflake*.

Quanto ao fato (Metric_Fact), este foi identificado por ser semi-aditivo, pois pode ser agregado em relação a dimensão tempo, resultando assim em agregações maiores para um mesmo projeto (meses, trimestres, semestres, anos).

O modelo da Figura 11 foi implementado no banco de dados MySQL Community ⁶ por meio da tradução do projeto lógico em projeto físico, que o próprio MySQL

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

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

Workbench realiza, caso o banco de dados destino seja o MySQL.

4.4 Ferramentas de DWing

Tendo em vista que o *Data Warehouse* foi projetado em um modelo dimensional, é possível construir tanto o processo de *Extraction-Transformation-Load* quanto as operações de consulta OLAP. Entre as alternativas de código aberto que suportam este ambiente como um todo, está o Pentaho BI Suite Community Edition. Este apresenta soluções que cobrem as áreas de ETL, *reporting*, OLAP e mineração de dados. Cada um dos componentes utilizados é apresentado e analisado nas seções subsequentes.

4.4.1 Implementação da Extração, Transformação e Carga dos Dados

O Pentaho Data Integration Community Edition ou Kettle⁷, como é conhecido pela comunidade que o desenvolve, é feito na linguagem Java e implementa o processo de ETL (Extração, Transformação e Carga de Dados). A interface do Kettle é mostrada na Figura 12 e as principais características do Kettle e a análise quanto aos critérios gerais de seleção de ferramentas são apresentadas na Tabela 16.

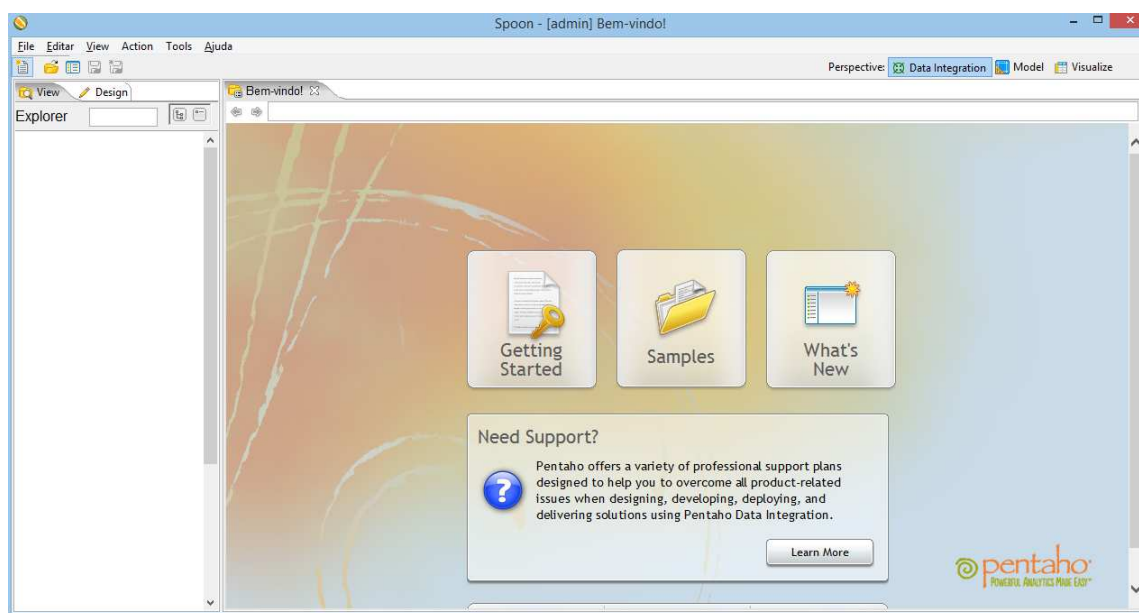


Figura 12 – Interface do Kettle

⁷ Disponível em <http://kettle.pentaho.com/>


Característica		CG01	CG02	CG03	CG04
Licença	Apache License 2.0	✓			
Integração com Banco de Dados	MySQL, SQLServer, PostgreSQL, Oracle entre outros				
Formatos Aceitos de Entrada de Dados	XML, TXT, JSON, ODS, XLS, CSV, Tabelas, YAML				
Ultima Versão Estável (14/11/2013)	4.4				✓
Quantidade de Commits no Repositório Oficial	10.000			✓	
Idioma da Documentação	Inglês		✓		
Quantidade de Casos Abertos no <i>Issue Tracker</i>	2875			✓	

Tabela 16 – Características do Kettle e avaliação quanto aos critérios gerais de seleção de ferramentas

O Kettle possui dois tipos de componentes internos: *Job* e *Transformation*. O primeiro permite executar tarefas, em nível mais alto, de fluxo de controle, tais como, mandar um email em caso de falha, baixar um arquivo, executar transformações e entre outras atividades. Já a *Transformation* permite tratamento aos dados incluindo desde entrada de dados por diversas fontes até a persistência em uma variedade de SGBDs.

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

4.4.2 Implementação das Consultas OLAP e Visualização de Dados

Para a implementação das consultas OLAP e Visualização de dados, torna-se necessário a utilização do Pentaho BI Platform⁸, que é uma ferramenta que provê a arquitetura e a infraestrutura para soluções de *Business Intelligence*, *Data Mining* e a camada de visualização de dados do *Data Warehouse*.

O Pentaho BI Platform, cuja interface inicial é apresentada na Figura 13, tem as principais características e a análise quanto aos critérios gerais de seleção de ferramentas são apresentadas na Tabela 17.

⁸ Disponível em http://community.pentaho.com/projects/bi_platform/


Característica		CG01	CG02	CG03	CG04
					
Licença	Apache License 2.0	✓			
Integração com Banco de Dados	MySQL, SQLServer, PostgreSQL, Oracle entre outros				
Linguagem em que foi desenvolvida	Java				
Ultima Versão Estável (14/11/2013)	4.8				✓
Quantidade de Commits no Repositório Oficial em 14/11/2013	3700			✓	
Idioma da Documentação	Inglês		✓		
Quantidade de Casos Abertos no <i>Issue Tracker</i>	1489			✓	

Tabela 17 – Características do Pentaho BI Platform e avaliação quanto aos critérios gerais de seleção de ferramentas

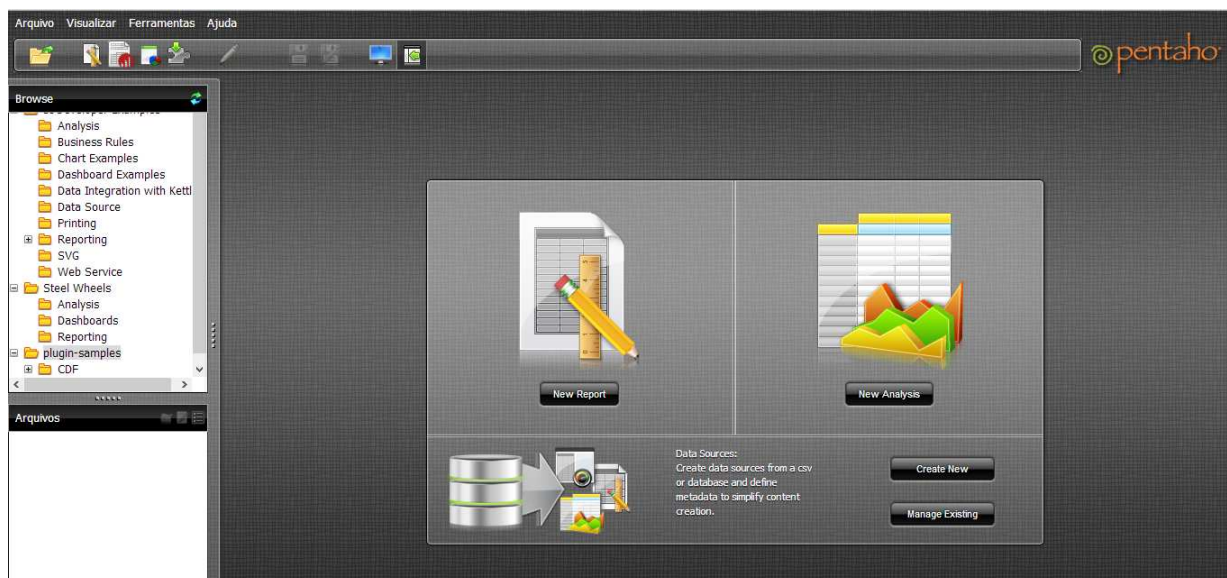


Figura 13 – Interface do Pentaho BI Platform

A ferramenta Pentaho BI Platform possui arquitetura extensível por plugins diversos que realizam diversas operações, tais como, criação de relatórios, visualização dos dados em tabelas e gráficos e entre outros. Entre os plugins disponíveis, está o Saiku Analytics que oferece serviços de apoio a operações OLAP e à visualização de dados. As

características gerais do Saiku Analytics, bem como a avaliação quanto aos critérios gerais de seleção de ferramentas, são apresentados na Tabela 18.


Característica		CG01	CG02	CG03	CG04
Licença	GPL 2.0	✓			
Componentes de Visualização	Tabelas e Gráficos				
Gráficos com Suporte	Gráfico de Pizza, Gráfico de Linhas, Gráfico de Área, Gráfico de Setor e entre outros				
Ultima Versão Estável (14/11/2013)	2.5				✓
Quantidade de Commits no Repositório Oficial em 14/11/2013	790			✓	
Idioma da Documentação	Inglês		✓		
Quantidade de Casos Abertos no <i>Issue Tracker</i>	227			✓	

Tabela 18 – Características do Saiku Analytics e avaliação quanto aos critérios gerais de seleção de ferramentas

O Saiku Analytics possui em sua arquitetura outro componente da arquitetura do Pentaho BI Suite, que é o Mondrian OLAP Server. Este permite ao Saiku, que sejam realizadas consultas *ad hoc* com as dimensões do cubo, de um esquema dimensional, realizando *drag and drop* das colunas das dimensões.

Por meio do Saiku, permite-se a realização de consultas. Estas ocorrem por meio da escrita de *queries* em linguagem MDX (*Multidimensional eXpressions*). Esta foi proposta por Spofford et al. (2006) como uma forma de escrever consultas mais otimizadas para bases seguem o modelo dimensional, tal como mostra o exemplo do trecho de Código-Fonte 1.

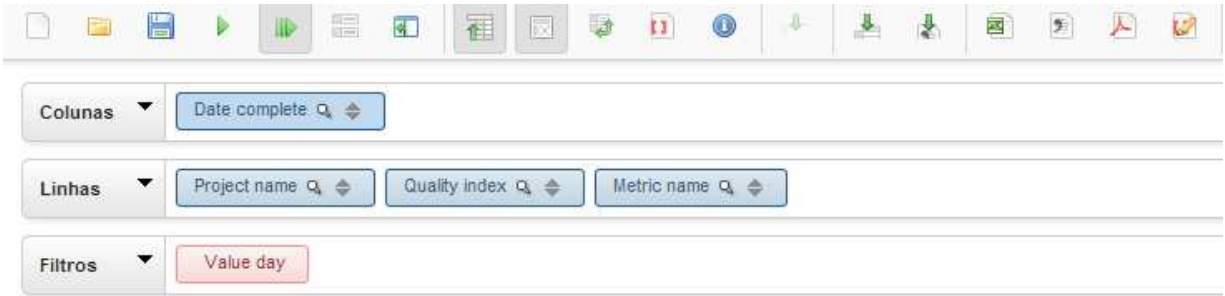
```
1  SELECT
2    { [Measures].[Loja] } ON COLUMNS,
3    { [Tempo].[2002], [Tempo].[2003]
      } ON ROWS
4  FROM Vendas
5  WHERE ( [Loja].[Loja Sul])
```

Código-Fonte 1 – Exemplo de *Query* em
linguagem MDX

5 Considerações Finais

5.1 Resultados Parciais

Após o acompanhamento das métricas de código-fonte do projeto *Apache Maven* do dia 07/09/2013 a 10/11/2013, obteve-se as Figura 16 e a Figura 15, resultantes da implementação de algumas consultas OLAP *ad hoc*.



Project name	Quality index	Metric name	2013-09-07	2013-10-13	2013-10-20	2013-10-27	2013-11-03	2013-11-10
Apache Maven	Bom	ACC	3	3	3	3	3	3
		LOC	40	40	40	40	40	40
		RFC	23	23	23	23	23	23
	Excelente	ACCM	3	2	2	2	2	2
		DIT	2	2	2	2	2	2
		LCOM4	1	1	1	1	1	1
		NOC	1	1	1	1	1	1
		NOM	8	8	8	8	8	8

Figura 14 – Visualização de dados do Apache Maven em formato de tabela no Saiku Analytics

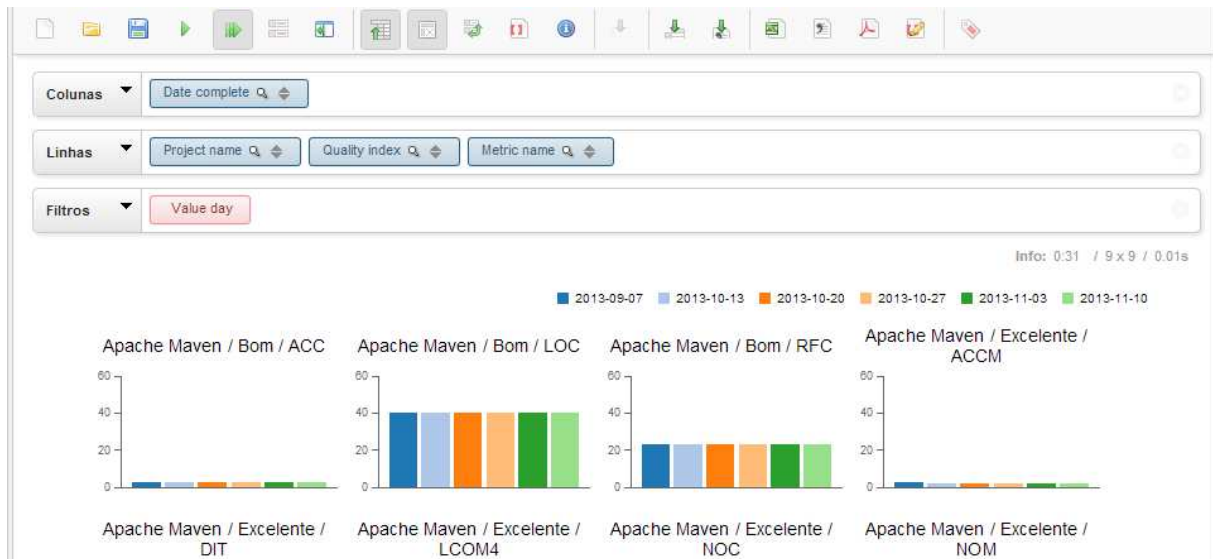


Figura 15 – Visualização de dados do Apache Maven em formato de gráfico de barras no Saiku Analytics

Após a implementação do ambiente de *Data Warehousing* e a realização do estudo de caso sobre o ambiente, verifica-se que a hipótese de pesquisa foi validada parcialmente. Isto é, o ambiente de DWIng possibilita maior poder de análise sobre as métricas de código-fonte. Isso ocorre porque a camada de visualização do ambiente de DWIng permite uma flexibilidade de consultas que não está disponível nas ferramentas de análise estática de código-fonte e disponibiliza a informação da qualidade do código-fonte em formas visuais (gráficos e tabelas) as quais facilitam a interpretação da informação.

A análise dos gráficos e tabelas isolados na camada de visualização ainda não é a melhor forma de transmitir os dados, logo a hipótese deve ser verificada também quando for implementado o *Dashboard*.

Com a realização do estudo de caso da coleta de dados, das métricas de código-fonte do Apache maven, observou-se que o comportamento de estabilidade das mesmas. Logo foi avaliado também a a média e mediana, como se mostra no apêndice A.

Após a análise de média e mediana e percentis, corrobora-se com [Meirelles \(2013\)](#), pois a análise das métricas de código-fonte utilizando os percentis mostra valores representativos da qualidade geral do código-fonte. Mostra-se ainda, por meio da análise dos dados, que o Apache Maven é um projeto maduro e que, segundo os dados coletados, apresenta as métricas ACC, LOC, RFC em nível bom e as métricas ACCM, DIT, LCOM4, NOC, NOM em nível excelente.

5.2 Trabalhos Futuros

Na próxima etapa deste trabalho, será i) melhor identificado outras demandas do negócio por meio de entrevistas semi-estruturadas com profissionais atuantes em métricas de código-fonte ii) investigado a utilização de pesos para cada métrica de código-fonte, possibilitando assim configurações diferenciadas para cada projeto; iii) avaliado as principais necessidades de informação em métricas de código-fonte, com objetivo de implementar o *Dashboard* como componente de visualização; iv) comparado o Pentaho com o SpargoBI, que também é uma outra ferramenta livre que possibilita o desenvolvimento do ambiente de DWing; v) implementado um estudo de caso em múltiplos projetos do ambiente de DWing. Com objetivo de distribuir as atividades de pesquisa e implementação, foi elaborado um calendário de marcos, tal como se vê na Figura

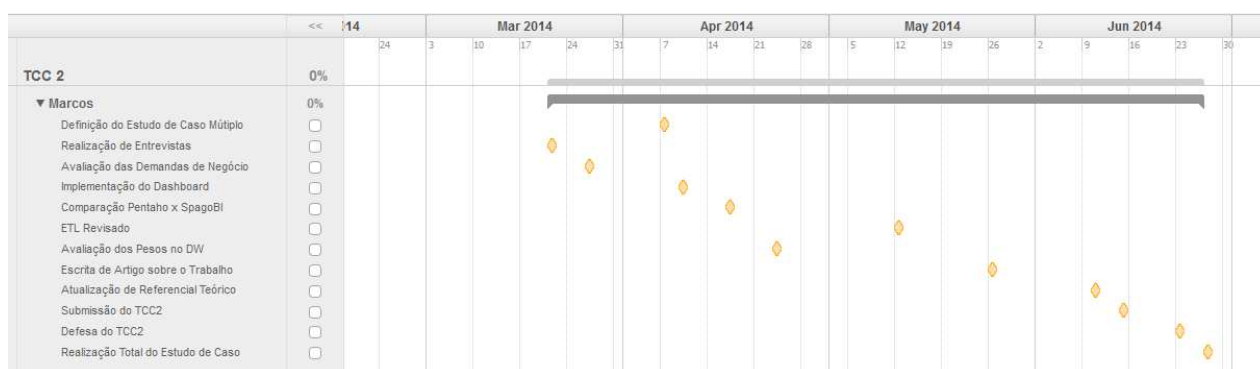


Figura 16 – Calendário de Marcos da próxima etapa deste Trabalho de Conclusão de Curso

Referências

- BASILI, V. R.; CALDIERA, G.; ROMBACH, H. D. *The Goal Question Metric Approach*. [S.l.]: Encyclopedia of Software Engineering, 1996. Citado na página 17.
- BECK, K. Embracing change with extreme programming. *Computer*, v. 32, n. 10, p. 70–77, 1999. ISSN 0018-9162. Citado na página 39.
- BECK, K. *Extreme Programming Explained*. [S.l.]: Addison Wesley, 1999. Citado 2 vezes nas páginas 14 e 38.
- BECK, K. *Implementation patterns*. [S.l.]: Pearson Education, 2007. Citado na página 14.
- BECKER, K. et al. Spdw: A software development process performance data warehousing environment. In: *Proceedings of the 30th Annual IEEE/NASA Software Engineering Workshop*. Washington, DC, USA: IEEE Computer Society, 2006. (SEW '06), p. 107–118. ISBN 0-7695-2624-1. Disponível em: <<http://dx.doi.org/10.1109/SEW.2006.31>>. Citado na página 15.
- CASTELLANOS, M. et al. ibom: A platform for intelligent business operation management. In: *Proceedings of the 21st International Conference on Data Engineering*. Washington, DC, USA: IEEE Computer Society, 2005. (ICDE '05), p. 1084–1095. ISBN 0-7695-2285-8. Disponível em: <<http://dx.doi.org/10.1109/ICDE.2005.73>>. Citado na página 15.
- CHAUDHURI, S.; DAYAL, U. An overview of data warehousing and olap technology. *ACM Sigmod record*, ACM, v. 26, n. 1, p. 65–74, 1997. Citado 2 vezes nas páginas 24 e 28.
- CHIDAMBER, S. R.; KEMERER, C. F. A Metrics Suite for Object-Oriented Design. *IEEE Transactions on Software Engineering*, v. 20, n. 6, p. 476–493, 1994. Citado na página 21.
- CHULANI, S. et al. Metrics for managing customer view of software quality. In: *Proceedings of the 9th International Symposium on Software Metrics*. Washington, DC, USA: IEEE Computer Society, 2003. (METRICS '03), p. 189–. ISBN 0-7695-1987-3. Disponível em: <<http://dl.acm.org/citation.cfm?id=942804.943748>>. Citado na página 15.
- CMMI. *CMMI® for Development, Version 1.3*. [S.l.], 2010. Disponível em: <<http://www.sei.cmu.edu/library/abstracts/reports/10tr033.cfm>>. Citado na página 17.
- CODD, E. F.; CODD, S. B.; SALLEY, C. T. *Providing OLAP (On-Line Analytical Processing) to User-Analysis: An IT Mandate*. [S.l.]: E. F. Codd & Associates, 1993. Citado na página 27.
- EMANUELSSON, P.; NILSSON, U. A comparative study of industrial static analysis tools. *Electron. Notes Theor. Comput. Sci.*, Elsevier Science Publishers B. V.,

Amsterdam, The Netherlands, The Netherlands, v. 217, p. 5–21, jul. 2008. ISSN 1571-0661. Citado na página 14.

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

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

FOLLECO, A. et al. Learning from software quality data with class imbalance and noise. In: *Proceedings of the Nineteenth International Conference on Software Engineering & Knowledge Engineering (SEKE 2007), Boston, Massachusetts, USA, July 9-11, 2007*. [S.l.]: Knowledge Systems Institute Graduate School, 2007. p. 487. ISBN 1-891706-20-9. Citado na página 15.

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

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

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

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

GOPAL, A.; MUKHOPADHYAY, T.; KRISHNAN, M. S. The impact of institutional forces on software metrics programs. *IEEE Trans. Softw. Eng.*, IEEE Press, Piscataway, NJ, USA, v. 31, n. 8, p. 679–694, ago. 2005. ISSN 0098-5589. Disponível em: <<http://dx.doi.org/10.1109/TSE.2005.95>>. Citado na página 24.

HARMAN, M. Why source code analysis and manipulation will always be important. In: *IEEE. Source Code Analysis and Manipulation (SCAM), 2010 10th IEEE Working Conference on*. [S.l.], 2010. p. 7–19. Citado na página 19.

HITZ, M.; MONTAZERI, B. Measuring Coupling and Cohesion in Object-Oriented Systems. In: *Proceedings of International Symposium on Applied Corporate Computing*. [S.l.: s.n.], 1995. Citado na página 21.

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

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

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

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

- KIMBALL, R.; ROSS, M. *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling*. 2nd. ed. New York, NY, USA: John Wiley & Sons, Inc., 2002. ISBN 0471200247, 9780471200246. Citado 6 vezes nas páginas 25, 26, 27, 29, 38 e 39.
- LEHMAN, M. M. Programs, life cycles, and laws of software evolution. *Proc. IEEE*, v. 68, n. 9, p. 1060–1076, September 1980. Citado 2 vezes nas páginas 20 e 37.
- LORENZ, M.; KIDD, J. *Object-Oriented Software Metrics*. [S.l.]: Prentice Hall, 1994. Citado na página 21.
- MCCABE, T. J. A Complexity Measure. *IEEE Transactions Software Engineering*, v. 2, n. 4, p. 308–320, December 1976. Citado na página 20.
- MCCABE, T. J.; DREYER, L. A.; WATSON, A. H. Testing An Object-Oriented Application. *Journal of the Quality Assurance Institute*, v. 8, n. 4, p. 21–27, October 1994. Citado na página 21.
- MEIRELLES, P. R. M. *Monitoramento de métricas de código-fonte em projetos de software livre*. Tese (Doutorado) — Instituto de Matemática e Estatística – Universidade de São Paulo (IME/USP), 2013. Citado 11 vezes nas páginas 14, 18, 20, 21, 22, 35, 36, 37, 40, 41 e 48.
- MILLS, E. E. Metrics in the software engineering curriculum. *Ann. Softw. Eng.*, J. C. Baltzer AG, Science Publishers, Red Bank, NJ, USA, v. 6, n. 1-4, p. 181–200, abr. 1999. ISSN 1022-7091. Citado 2 vezes nas páginas 18 e 20.
- MINARDI, R. C. de M. *Visualização de Dados*. 2013. Universidade Federal de Minas Gerais - UFMG. Disponível em: <<http://homepages.dcc.ufmg.br/~raquelcm/material/visualizacao/aulas/>>. Citado 3 vezes nas páginas 9, 31 e 32.
- NERI, H. R. *Análise, Projeto e Implementação de um Esquema MOLAP de Data Warehouse utilizando SGBD-OR Oracle 8.1*. Universidade Federal da Paraíba - UFPB: [s.n.], 2002. Citado 2 vezes nas páginas 10 e 28.
- NIELSON, F.; NIELSON, H. R.; HANKIN, C. *Principles of Program Analysis*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1999. ISBN 3540654100. Citado na página 14.
- PALZA, E.; FUHRMAN, C.; ABRAN, A. Establishing a generic and multidimensional measurement repository in cmmi context. In: IEEE. *Software Engineering Workshop, 2003. Proceedings. 28th Annual NASA Goddard*. [S.l.], 2003. p. 12–20. Citado na página 15.
- ROCHA, A. B. *Guardando Históricos de Dimensões em Data Warehouses*. Universidade Federal da Paraíba - Centro de Ciências e Tecnologia: [s.n.], 2000. Citado 6 vezes nas páginas 9, 10, 24, 28, 29 e 30.
- ROSENBERG, L. H.; HYATT, L. E. Software Quality Metrics for Object-Oriented Environments. *Crosstalk - the Journal of Defense Software Engineering*, v. 10, 1997. Citado na página 21.

- RUIZ, D. D. A. et al. A data warehousing environment to monitor metrics in software development processes. In: *16th International Workshop on Database and Expert Systems Applications (DEXA 2005)*, 22-26 August 2005, Copenhagen, Denmark. [S.l.]: IEEE Computer Society, 2005. p. 936–940. ISBN 0-7695-2424-9. Citado na página 15.
- SHARBLE, R.; COHEN, S. The Object-Oriented Brewery: A Comparison of Two Object-Oriented Development Methods. *Software Engineering Notes*, v. 18, n. 2, p. 60–73, 1993. Citado na página 21.
- SHIH, T. et al. Decomposition of Inheritance Hierarchy DAGs for Object-Oriented Software Metrics. In: *Workshop on Engineering of Computer-Based Systems (ECBS 97)*. [S.l.: s.n.], 1997. p. 238. Citado na página 21.
- SILVEIRA, P. S.; BECKER, K.; RUIZ, D. D. Spdw+: a seamless approach for capturing quality metrics in software development environments. *Software Quality Control*, Kluwer Academic Publishers, Hingham, MA, USA, v. 18, n. 2, p. 227–268, jun. 2010. ISSN 0963-9314. Disponível em: <<http://dx.doi.org/10.1007/s11219-009-9092-9>>. Citado 2 vezes nas páginas 15 e 24.
- SOMMERVILLE, I. *Software Engineering*. 9. ed. Harlow, England: Addison-Wesley, 2010. ISBN 978-0-13-703515-1. Citado na página 14.
- SPOFFORD, G. et al. *MDX Solutions with Microsoft SQL Server Analysis Services 2005 and Hyperion Essbase*. [S.l.]: Wiley Pub., 2006. Citado na página 45.
- TERRA, R.; BIGONHA, R. S. Ferramentas para análise estática de códigos java. Departamento de Ciência da Computação - UFMG, 2008. Citado na página 14.
- TIMES, V. C. *Sistemas de DW*. 2012. Universidade Federal de Pernambuco - UFPE. Disponível em: <www.cin.ufpe.br/~if695/bda_dw.pdf>. Citado 6 vezes nas páginas 9, 10, 26, 27, 28 e 30.
- WICHMANN, B. et al. Industrial perspective on static analysis. *Software Engineering Journal*, 1995. Citado na página 14.

APÊNDICE A – Métricas de Código-Fonte do Apache Maven

Data da Análise	Métrica	Média	Mediana	Percentil
07/09/2013	LOC	82,7	40	40
	ACCM	2	1,3	2,7
	ACC	2,6	1	3
	RFC	21	9	23
	LCOM4	1,1	1	1
	NOM	7,2	4	8
	DIT	1,2	1	2
	NOC	0,4	0	1

Tabela 19 – Métricas de Código Fonte do Apache Maven em 07/09/2013

Data da Análise	Métrica	Média	Mediana	Percentil
13/10/2013	LOC	89,5	40	40
	ACCM	1,8	1	2
	ACC	2,6	1	3
	RFC	20,8	8	23
	LCOM4	1,1	1	1
	NOM	7,2	4	8
	DIT	1,2	1	2
	NOC	0,4	0	1

Tabela 20 – Métricas de Código Fonte do Apache Maven em 13/10/2013

Data da Análise	Métrica	Média	Mediana	Percentil
20/10/2013	LOC	89,5	40	40
	ACCM	1,8	2	2
	ACC	2,6	1	3
	RFC	20,8	8	23
	LCOM4	1,1	1	1
	NOM	7,2	4	8
	DIT	1,2	1	2
	NOC	0,4	0	1

Tabela 21 – Métricas de Código Fonte do Apache Maven em 20/10/2013

Data da Análise	Métrica	Média	Mediana	Percentil
27/10/2013	LOC	89,5	40	40
	ACCM	1,8	2	2
	ACC	2,6	1	3
	RFC	20,8	8	23
	LCOM4	1,1	1	1
	NOM	7,2	4	8
	DIT	1,2	1	2
	NOC	0,4	0	1

Tabela 22 – Métricas de Código Fonte do Apache Maven em 27/10/2013

Data da Análise	Métrica	Média	Mediana	Percentil
03/11/2013	LOC	89,5	40	40
	ACCM	1,8	2	2
	ACC	2,6	1	3
	RFC	20,8	8	23
	LCOM4	1,1	1	1
	NOM	7,2	4	8
	DIT	1,2	1	2
	NOC	0,4	0	1

Tabela 23 – Métricas de Código Fonte do Apache Maven em 03/11/2013

Data da Análise	Métrica	Média	Mediana	Percentil
10/11/2013	LOC	89,5	40	40
	ACCM	1,8	2	2
	ACC	2,6	1	3
	RFC	20,8	8	23
	LCOM4	1,1	1	1
	NOM	7,2	4	8
	DIT	1,2	1	2
	NOC	0,4	0	1

Tabela 24 – Métricas de Código Fonte do Apache Maven em 10/11/2013

APÊNDICE B – Descrição Simplificada do Processo de ETL no Kettle

No Kettle, o processo de ETL foi modelado de forma a obter a métricas do código-fonte do projeto. Para isso, executa-se um Job, que é o componente de fluxo de controle tal como se mostra na Figura 17.

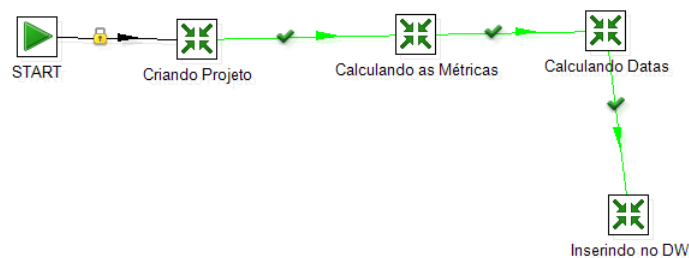


Figura 17 – *Job* no Kettle

Esse *Job* envolve uma série de transformações para que se alcance o resultado final, que são as métricas de código-fonte agregadas em nível do projeto no *Data Warehouse*.

A primeira *Trasformation* envolve gravar os dados do projeto, que advém de um arquivo JSON extraído do SonarQube no DW, por meio dos componentes do Kettle, tal como se mostra na Figura 18.



Figura 18 – Primeira *Trasformation* do ETL

A segunda *Trasformation*, que tem nome de "Calculando as Métricas", é a principal *Trasformation* do ETL. O fluxo desta é demonstrado na Figura 19,

Figura 19 – Segunda *Transformation* do ETL

O processamento da segunda *Transformation* inicia-se com a leitura de um arquivo JSON semelhante ao expresso abaixo:

```

1  [{
2    "id": 115771,
3    "key": "org.apache.maven:maven-compat:org.apache.maven.artifact.
      repository.layout.FlatRepositoryLayout",
4    "name": "FlatRepositoryLayout",
5    "scope": "FIL",
6    "qualifier": "CLA",
7    "date": "2013-11-03T00:49:39+0100",
8    "creationDate": null,
9    "lname": "org.apache.maven.artifact.repository.layout.
      FlatRepositoryLayout",
10   "lang": "java",
11   "msr": [{
12     "key": "ncloc",
13     "val": 51.0,
14     "frmt_val": "51"
15   }, {
16     "key": "functions",
17     "val": 6.0,
18     "frmt_val": "6"
19   }, {
20     "key": "function_complexity",
21     "val": 1.5,
22     "frmt_val": "1,5"
23   }, {
24     "key": "dit",
25     "val": 1.0,
26     "frmt_val": "1"
27   }, {
28     "key": "noc",
29     "val": 0.0,
30     "frmt_val": "0"
31   }, {
32     "key": "rfc",
33     "val": 21.0,

```

```

34     "frmt_val": "21"
35   }, {
36     "key": "lcom4",
37     "val": 2.0,
38     "frmt_val": "2,0"
39   }, {
40     "key": "ca",
41     "val": 0.0,
42     "frmt_val": "0"
43   }]
44 }]
```

Código-Fonte 2 – Exemplo de Métricas de uma classe em JSON

O processamento do JSON é realizado, então, com o componente *JSON input* do Kettle, tal como se mostra a Figura 20.

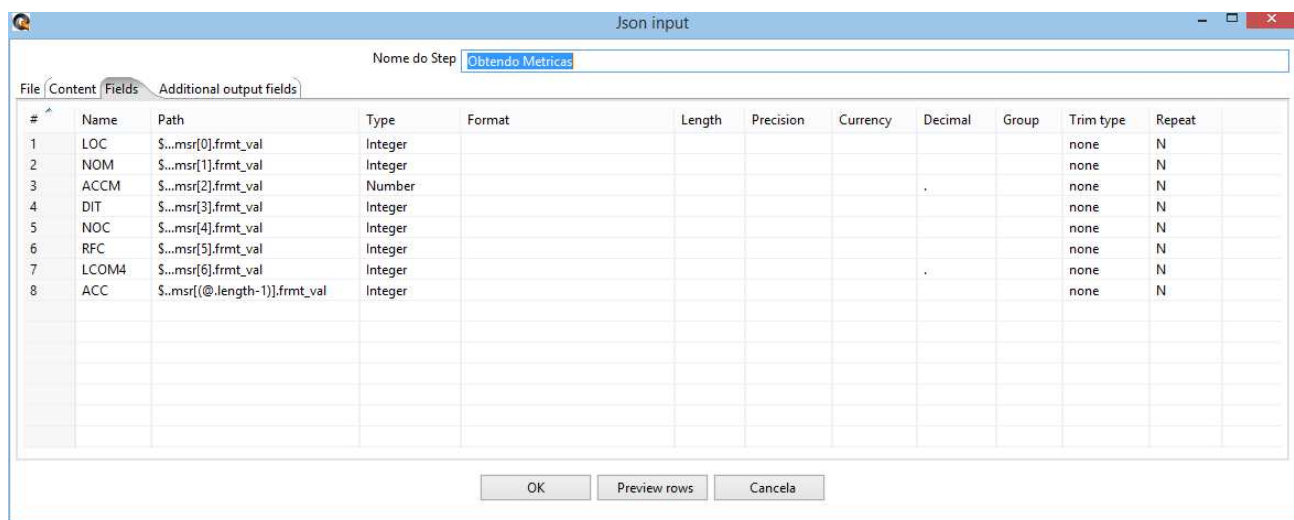
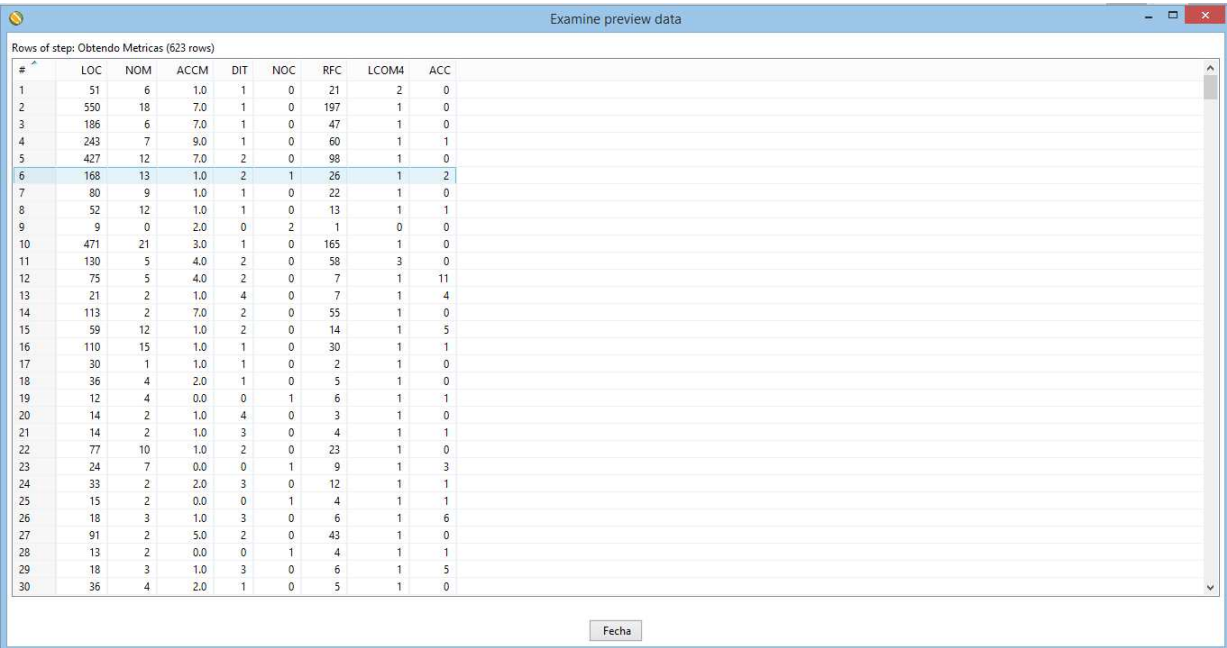


Figura 20 – Obtendo os Dados do JSON

Este resulta na obtenção, no ambiente do Kettle, das métricas de código-fonte que serão agregadas para se obter os valores dos percentis de cada métrica, tal como se vê na Figura 21.



Examine preview data

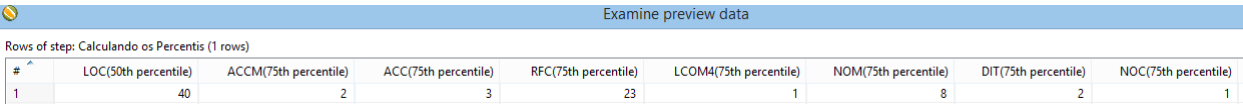
Rows of step: Obtendo Metricas (623 rows)

#	LOC	NOM	ACCM	DIT	NOC	RFC	LCOM4	ACC
1	51	6	1.0	1	0	21	2	0
2	550	18	7.0	1	0	197	1	0
3	186	6	7.0	1	0	47	1	0
4	243	7	9.0	1	0	60	1	1
5	427	12	7.0	2	0	98	1	0
6	168	13	1.0	2	1	26	1	2
7	80	9	1.0	1	0	22	1	0
8	52	12	1.0	1	0	13	1	1
9	9	0	2.0	0	2	1	0	0
10	471	21	3.0	1	0	165	1	0
11	130	5	4.0	2	0	58	3	0
12	75	5	4.0	2	0	7	1	11
13	21	2	1.0	4	0	7	1	4
14	113	2	7.0	2	0	55	1	0
15	59	12	1.0	2	0	14	1	5
16	110	15	1.0	1	0	30	1	1
17	30	1	1.0	1	0	2	1	0
18	36	4	2.0	1	0	5	1	0
19	12	4	0.0	0	1	6	1	1
20	14	2	1.0	4	0	3	1	0
21	14	2	1.0	3	0	4	1	1
22	77	10	1.0	2	0	23	1	0
23	24	7	0.0	0	1	9	1	3
24	33	2	2.0	3	0	12	1	1
25	15	2	0.0	0	1	4	1	1
26	18	3	1.0	3	0	6	1	6
27	91	2	5.0	2	0	43	1	0
28	13	2	0.0	0	1	4	1	1
29	18	3	1.0	3	0	6	1	5
30	36	4	2.0	1	0	5	1	0

Fechar

Figura 21 – Resultado do Processamento do *JSON input*

Após a realização da agregação estatística com os percentis obtém-se um resultado semelhante a Figura 22.



Examine preview data

Rows of step: Calculando os Percentis (1 rows)

#	LOC(50th percentile)	ACCM(75th percentile)	ACC(75th percentile)	RFC(75th percentile)	LCOM4(75th percentile)	NOM(75th percentile)	DIT(75th percentile)	NOC(75th percentile)
1	40	2	3	23	1	8	2	1

Figura 22 – Resultado do Processamento Estatístico das Métricas de Código-Fonte

A terceira *Transformation* envolve a criação de registro na dimensão Tempo tal como se mostra na Figura 23.

Figura 23 – Terceira *Transformation* do ETL

Por fim a quarta *Transformation* realiza a partir do arquivo de propriedades no qual foi gravado todos os registros das dimensões, a carga do fatos do DW tal como se mostra na Figura 24.

Figura 24 – Quarta *Transformation* do ETL

Todos os arquivos das transformações, jobs e etc podem ser encontrados, com detalhes, no repositório Github: <https://github.com/gbreago/TCC>.