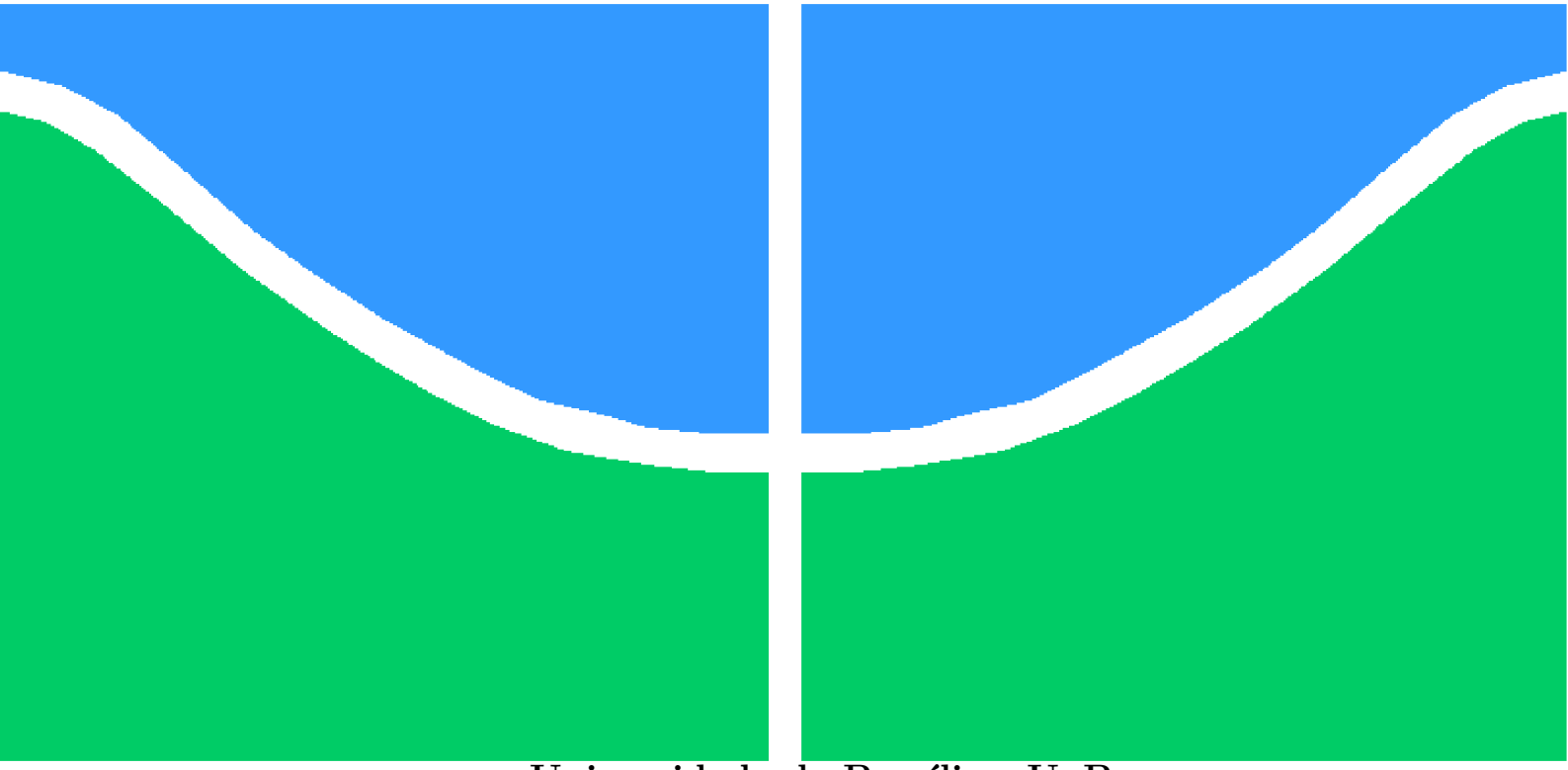

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Engenharia de Software

Estudo da Eficácia e Eficiência do Uso de um Ambiente de Data *Warehousing* para Aferição da Qualidade Interna de *Software*: Um Estudo de Caso em uma Autarquia Pública Federal

Autor: Nilton Cesar Campos Araruna

Orientador: Prof. Msc. Hilmer Rodrigues Neri

Brasília, DF

2014

Nilton Cesar Campos Araruna

Estudo da Eficácia e Eficiência do Uso de um Ambiente de Data *Warehousing* para Aferição da Qualidade Interna de *Software*: Um Estudo de Caso em uma Autarquia Pública Federal

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

Brasília, DF

2014

Nilton Cesar Campos Araruna

Estudo da Eficácia e Eficiência do Uso de um Ambiente de Data *Warehousing* para Aferição da Qualidade Interna de *Software*: Um Estudo de Caso em uma Autarquia Pública Federal/ Nilton Cesar Campos Araruna. – Brasília, DF, 2014-

50 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Msc. Hilmer Rodrigues Neri

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2014.

1. Métricas de Código-Fonte. 2. *Data Warehousing*. I. Prof. Msc. Hilmer Rodrigues Neri. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Estudo da Eficácia e Eficiência do Uso de um Ambiente de Data *Warehousing* para Aferição da Qualidade Interna de *Software*: Um Estudo de Caso em uma Autarquia Pública Federal

CDU a obter

Nilton Cesar Campos Araruna

Estudo da Eficácia e Eficiência do Uso de um Ambiente de Data *Warehousing* para Aferição da Qualidade Interna de *Software*: Um Estudo de Caso em uma Autarquia Pública Federal

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, a obter:

Prof. Msc. Hilmer Rodrigues Neri
Orientador

---

a obter
Convidado 1

---

a obter
Convidado 2

Brasília, DF
2014

Este trabalho é dedicado ao meu pai, Aurenilton Araruna, minha mãe, Cláudia Araruna e ao meu irmão, Gustavo Campos. Estas inseriram na minha vida as virtudes do esforço.

Agradecimentos

Agradeço ao 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 pelas oportunidades concedidas no projeto Desafio Positivo e no SRA, quanto pela orientação no presente trabalho.

Agradeço também ao meu coorientador, Prof. Paulo Meirelles por ter me mostrado o mundo do software livre, por ter compartilhado conhecimento sobre Métricas de Código-Fonte, Git, Ruby, Rails e em especial por ter compartilhado suas ponderações muito importantes para o meu crescimento pessoal e profissional.

Agradeço a Deus, inteligência suprema criadora de todas as coisas, por cada dia que é uma nova chance no caminho da evolução. Também 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 ainda 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 maior parte de minha criação, devo a ela desde todo o suporte fornecido em casa até a preocupação sobre a quantidade de horas que estava dormindo durante a noite.

Agradeço a Luisa Helena Lemos da Cruz por ser tão comprensiva, paciente, amorosa e dedicada para comigo, sobretudo nos dias que antencederam a entrega deste trabalho. A ela devo todas as transformações positivas de minha vida desde setembro 2012 até então. Sem sombras de dúvidas, ela é minha fonte de dedicação e 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 todo apoio dos grandes amigos: Gustavo Nobre Dias, Henrique Leão de Sá Menezes, Danilo Ribeiro Tosta e Carlos Henrique Lima Pontes.

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 também aos companheiros de Engenharia de Software: Vinícius Vieira Meneses, Matheus Freire, Renan Costa, Luiz Mattos, Pedro Tomioka, José Pedro Santana, Aline Gonçalves, Alexandre Almeida, Lucas Kanashiro, Fábio Texeira, Thiago Ribeiro, Alessandro Cateano, Gabriel Silva, Thaiane Braga, Maxwell Almeida e Carlos Oliviera.

"Learn from yesterday, live for today, hope for tomorrow. The important thing is not to stop questioning."

(Albert Einstein)

Resumo

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

Abstract

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

Lista de ilustrações

Figura 1 – Metodologia de Pesquisa 17
Figura 2 – Passos do Estudo de Caso 18
Figura 3 – Modelo de Qualidade do Produto da ISO/IEC 9126 (2001) 22
Figura 4 – Escopo do Estudo de Caso 40
Figura 5 – Estrutura do Estudo de Caso 41

Lista de tabelas

Tabela 1 – Percentis para métrica NOC em projetos Java extraídos de Meirelles (2013) 26
Tabela 2 – Nome dos Intervalos de Frequência extraídos de Rêgo (2014) 27
Tabela 3 – Configurações para os Intervalos das Métricas para Java extraídas de Rêgo (2014) 28
Tabela 4 – Conceitos de Limpeza levantados por Machini et al. (2010) e adaptados por Rêgo (201
Tabela 5 – Conceitos de Limpeza levantados por Machini et al. (2010) e adaptados por Rêgo (201
Tabela 6 – Conceitos de Limpeza levantados por Machini et al. (2010) e adaptados por Rêgo (201
Tabela 7 – Conceitos de Limpeza levantados por Machini et al. (2010) e adaptados por Rêgo (201
Tabela 8 – Cenários de Limpeza extraídos de Rêgo (2014) 35

Lista de abreviaturas e siglas

ACC	*Afferent Connections per Class*
ACCM	*Average Cyclomatic Complexity per Method*
AMLOC	*Average Method Lines of Code*
ANPM	*Average Number of Parameters per Method*
CBO	*Coupling Between Objects*
CSV	*Comma-Separated Values*
DER	Diagrama Entidade Relacionamento
DIT	*Depth of Inheritance Tree*
DW	*Data Warehouse*
ETL	*Extraction-Transformation-Load*
FTP	*File Transfer Protocol*
GQM	*Goal-Question-Metric*
IEC	*International Electrotechnical Commission*
ISO	*International Organization for Standardization*
JSON	*JavaScript Object Notation*
LCOM4	*Lack of Cohesion in Methods*
LOC	*Lines of Code*
NPA	*Number of Public Attributes*
NOC	*Number of Children*
NOM	*Number of Methods*
OLAP	*On-Line Analytical Processing*
OLTP	*Online Transaction Processing*
RFC	*Response For a Class*

SCAM	*IEEE International Working Conference on Source Code Analysis and Manipulation*
SGBD	Sistema de Gerenciamento de Bancos de Dados
SICG	Sistema Integrado de Conhecimento e Gestão
XML	*Extensible Markup Language*
YAML	*YAML Ain't Markup Language*

Sumário

1 INTRODUÇÃO 15
1.1 Contexto 15
1.2 Justificativa 15
1.3 Problema 16
1.4 Objetivos 16
1.5 Metodologia de pesquisa 17
1.6 Organização do Trabalho 19

2 MÉTRICAS DE SOFTWARE 21
2.1 Processo de Medição 21
2.2 Definição das métricas de software 21
2.3 Métricas de código fonte 23
2.3.1 Métricas de tamanho e complexidade 23
2.3.2 Métricas de Orientação a Objetos 24
2.4 Configurações de qualidade para métricas de código fonte 25
2.5 Cenários de limpeza 29
2.6 Considerações Finais do Capítulo 36

3 CONTRATAÇÕES DE FORNECEDORES DE DESENVOLVIMENTO DE SOFTW
3.1 Importância da Contratação de Fornecedores de Desenvolvimento de Software 37
3.2 Normas, Processos e Legislação Pertinentes à Contratação 37
3.3 Contrato do Órgão X 37

4 *DATA WAREHOUSE* 38
4.1 Ciclo de vida de um *Data Warehousing* 38
4.1.1 *Extraction-Transformation-Load* 38
4.2 Modelagem Dimensional 38
4.2.1 OLAP *On-Line Analytic Processing* 38

5 PROJETO DE ESTUDO DE CASO 39
5.1 Definição 39
5.1.1 Problema 40
5.1.2 Questão de Pesquisa 40
5.2 *Background* 43
5.3 Seleção 44
5.4 Fonte dos dados coletados e método de coleta 44

5.5 Ameaças a validade do estudo de caso 44
5.6 Processo de análise dos dados 45
5.7 Considerações finais do capítulo 46

6 CONCLUSÃO 47

Referências 48

APÊNDICE A – DESCRIÇÃO DO PROCESSO DE ETL NO KETTLE 50

1 Introdução

1.1 Contexto

Segundo(WILLCOCKS; LACITY, 2001) a terceirização é uma prática cada vez mais adotada por organizações. A terceirização de atividades, ou seja, o ato de transferir para fora da organização uma parte do seu processo produtivo, não é uma prática recente, desde há muitos anos que as atividades e processos que eram muito específicos dentro de uma organização foram transferidos, de uma forma parcial ou total, para outras organizações ou agentes externo (LEITE, 1997).

Uma das motivações para a terceirização é a qualidade do serviço prometida pelas empresas fornecedoras. No entanto, existem vários riscos associados à decisão pela terceirização, que podem comprometer a qualidade esperada, como: as expectativas de serviço e a resposta rápida não serem atendidas adequadamente; o serviço prestado apresentar qualidade inferior ao existente anteriormente; e as tecnologias utilizadas não corresponderem ao esperado(WILLCOCKS; LACITY, 2001). Segundo o (SOFTEX, 2013a) a aquisição de um *software* é um processo complexo, principalmente no que diz respeito à caracterização dos requisitos necessários ao *software* e às condições envolvidas na contratação como a qualidade esperada.

Acompanhando o ritmo da terceirização o cenário com empresas terceirizadas contratadas para o desenvolvimento de *software* está cada vez maior em organizações públicas. Tais organizações não são diretamente responsáveis pelo desenvolvimento do *software* mas são responsáveis pelo processo de verificação de sua qualidade conforme a norma Brasil (2014) que será explicada no capítulo 3 deste trabalho.

1.2 Justificativa

Segundo (BECK, 1999)(FOWLER, 1999) a qualidade de *software* é medida pela qualidade de seu código-fonte. Conforme a (ISO/IEC 15939, 2002) medição é uma ferramenta primordial para avaliar a qualidade dos produtos e a capacidade de processos organizacionais, portanto o Órgão Público contratante pode fazer uso do monitoramento de métricas de código-fonte para assistir ao processo de aferição de qualidade do *software* desenvolvido pela contratada.

(RêGO, 2014) propôs uma solução para o monitoramento de métricas de código-fonte com suporte de um ambiente de *Data Warehousing* que será explicada no capítulo 5. Uma das principais contribuições desse trabalho será evidenciar a eficácia e a eficiência

da proposta por Rêgo (2014) no aferimento de qualidade do software adquirido de uma empresa terceirizada para equipe responsável do Órgão público e seus demais envolvidos.

1.3 Problema

Com foco na avaliação de controles gerais de Tecnologia da Informação nos órgãos públicos, em 2011, o TCU detectou, por meio de auditorias de governança de TI, em diversos órgãos uma considerável frequência de irregularidades relacionadas à inexistência, deficiências e a falhas de processos de *software* que comprometem a eficácia e eficiência das contratações de desenvolvimento de sistemas (BRASIL, 2011).

A inexistência de parâmetros de aferição de qualidade para contratação de desenvolvimento de sistemas e a deficiência no processo de contratação, decorrente da inexistência de metodologia que assegure boa contratação de desenvolvimento de sistemas foram listados nas auditorias como consequências da inexistência e falha de processos de *software* nos órgãos públicos. Os critérios indicados pelo TCU no acórdão (BRASIL, 2011) foram:Constituição Federal, art. 37, caput; Instrução Normativa 4/2008, SLTI/MPOG, art. 12, inciso II; Lei 8666/1993, art. 6º, inciso IX; Norma Técnica - ITGI - Cobit 4.1, PO8.3 - Padrões de desenvolvimento e de aquisições; Norma Técnica - NBR ISO/IEC - 12.207 e 15.504;e Resolução 90/2009, CNJ, art. 10.

A auditoria do acordão (BRASIL, 2011) reforça o problema da falta de capacidade da administração pública de aferir a qualidade interna dos produtos de software desenvolvido por terceirizadas. A partir desse problema e da solução para monitoramento de métricas de código-fonte com suporte de um ambiente de *Data Warehousing* proposta por Rêgo (2014) , foi formulada a questão de pesquisa geral deste trabalho que é:

O uso de DW para o monitoramento de métricas de código fonte para assistir ao processo de aferição de qualidade interna dos produtos de software desenvolvido por terceirizadas, do ponto de vista da equipe de qualidade no desenvolvimento de software na CAIXA, é eficaz e eficiente?

1.4 Objetivos

O objetivo geral deste trabalho é realizar um estudo de caso onde será analisado a eficácia e a eficiência no uso de DW para o monitoramento de métricas de código fonte para assistir ao processo de aferição de qualidade interna dos produtos de software desenvolvido por uma empresa terceirizada para uma organização pública brasileira a partir das métricas e cenários de limpeza coletados pelo código-fonte do *software* desenvolvido, de questionário aos principais envolvidos no processo de verificação de qualidade e da observação em campo. Dentre os objetivos específicos deste trabalho estão:

- Avaliar a eficácia e eficiência da Solução de DW no processo de aferição de qualidade interna dos produtos de software desenvolvido por terceirizadas.
- Definir, projetar e caracterizar o estudo de caso.
- Coletar métricas e cenários de limpeza a partir código-fonte do *software* adquirido pela organização selecionada;
- Realizar análise dos dados coletados.
- Relatar os resultados obtidos.

1.5 Metodologia de pesquisa

Nessa seção apresenta-se a metodologia de pesquisa adotada neste trabalho. Para isso, foram definidos: a natureza da pesquisa; o tipo de metodologia de pesquisa; o tipo de abordagem de pesquisa; os métodos de procedimentos de pesquisa e os tipos de técnicas de coletas de dados.

Os procedimentos de pesquisa selecionados foram pesquisa bibliográfica, documental, levantamento e estudo de caso. As técnicas de coleta de dados selecionadas foram entrevistas, questionários e registro de observação na vida real. A seleção metodológica é apresentada na Figura 1.

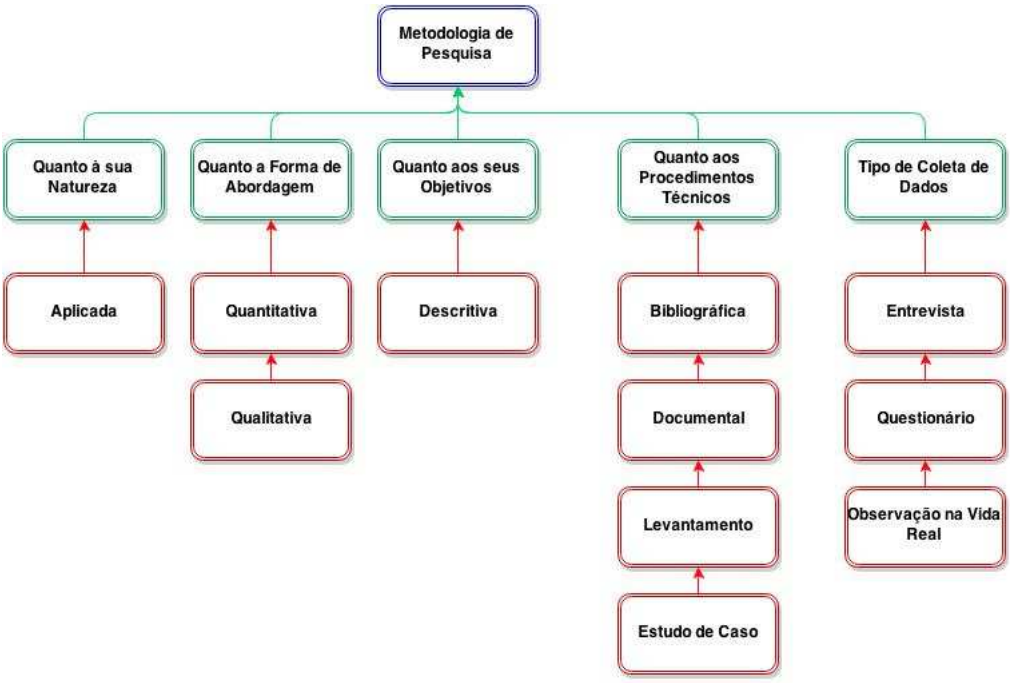

Figura 1 – Metodologia de Pesquisa

Segundo (YIN, 2001) o estudo de caso é um conjunto de procedimentos pré-especificados para se obter uma investigação empírica que investiga um fenômeno con-

temporâneo dentro de seu contexto da vida real, especialmente quando os limites entre o fenômeno e o contexto não estão claramente definidos. Uma grande vantagem do estudo de caso é a sua capacidade de lidar com uma ampla variedade de evidências - documentos, artefatos, entrevistas e observações - além do que pode estar disponível no estudo histórico convencional. Além disso, em algumas situações, como na observação participante, pode ocorrer manipulação informal.

A engenharia de *software* envolve o processo, desenvolvimento, operação e manutenção de *software* e artefatos relacionados. A maior parte da pesquisas na engenharia de software teve como objetivo investigar como o processo, desenvolvimento, operação e manutenção são conduzidos por engenheiros de *software* e outras partes interessadas em diferentes condições. Indivíduos, grupos e organizações, questões sociais e políticas são importantes para esse desenvolvimento. Isto é, a engenharia *software* é uma disciplina multidisciplinar que envolve áreas onde os estudos de caso são realizadas, como na psicologia, sociologia, ciência política, serviço social. Isto significa que muitas questões de pesquisa na engenharia de software são adequados para estudo de caso.(WOHLIN et al., 2012).

(WOHLIN et al., 2012) fraciona o estudo de caso em cinco passos. Nesta pesquisa, o estudo de caso compreende os passos: Planejar o Estudo de Caso; Coletar Dados; Analisar Dados Coletados e Compartilhar os Resultados. Portanto, os passos Projetar o Estudo de Caso e Preparar a Coleta de Dados definidas por Wohlin et al. (2012) foram agrupadas no passo Planejar o estudo de caso como na Figura 2.

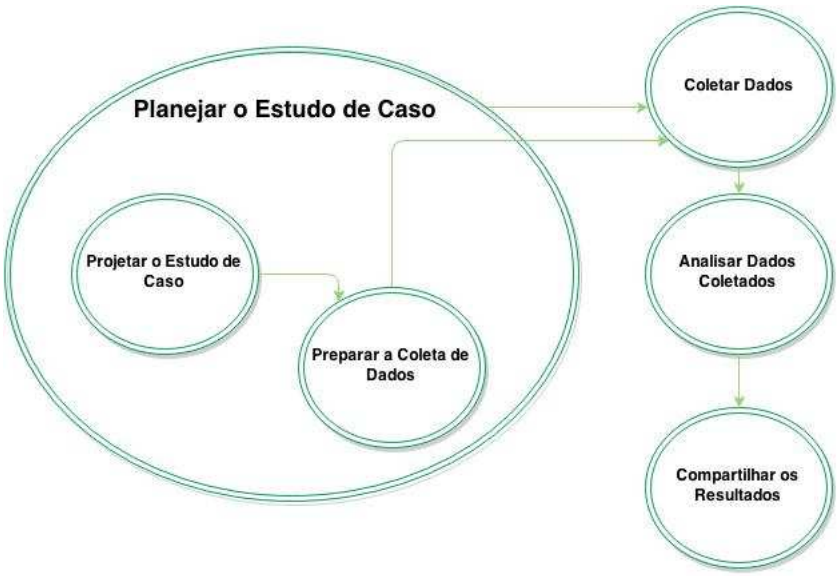

Figura 2 – Passos do Estudo de Caso

O passo Planejar o Estudo de Caso consiste na determinação do objetivo e da questão de pesquisa, da escolha da metodologia de pesquisa, da definição das fases da pesquisa, da definição do procedimentos de pesquisa, do protocolo, das técnicas de coleta

de dados e da proposta do trabalho final.

No passo Coletar Dados são executados os procedimentos de pesquisa e as técnicas de coletas de dados a seguir:

- Pesquisa Bibliográfica: pesquisa realizada a partir de livros, dissertações e trabalhos relacionados à área de pesquisa;
- Pesquisa Documental: pesquisa realizada a partir de documentos publicados por organizações públicas;
- Estudo de Caso: utilizar um estudo de caso real de uma organização pública brasileira;
- Entrevistas: dados serão coletados por meio de entrevistas informais, além de questionário, para incremento do estudo de caso;
- Documentos: coleta de dados dos documentos dos processos fornecidos pelo órgão público do estudo de caso será realizada para coleta de dados para análise;
- Observação na Vida Real: coleta de dados a partir da observação no campo de estudo;

O passo Analizar Dados Coletados d é onde os dados coletados serão analisados e interpretados. A análise compreende tanto a análise quantitativa quanto a análise qualitativa.

Por fim o passo Compartilhar os resultados diz respeito expor os resultados de forma adequada para o leitor alvo.

1.6 Organização do Trabalho

Esse trabalho está dividido em 5 capítulos:

- **Capítulo 1 - Introdução:** Esse capítulo tem como objetivo apresentar o contexto que esse trabalho está inserido, o problema sobre o qual ele buscará resolver, qual a justificativa, os objetivos da sua realização e metodologia de pesquisa adotada.
- **Capítulo 2 - Métricas de Software:** Capítulo responsável pela explicação teórica a respeito do que são métricas de código e como elas foram utilizadas no desenvolvimento da solução que esse trabalho busca analisar.
- **Capítulo 3 - Contratações de Fornecedores de Desenvolvimento de Software:** apresenta-se as principais informações referentes à Contratação de Fornecedores de Desenvolvimento de Software. Para isso, o capítulo é iniciado com uma visão geral sobre a importância das contratações e suas principais características. Poste-

riormente, é apresentado um resumo das legislações relacionadas à Contratação de Soluções de Tecnologia da Informação.

- **Capítulo 4 - Data Warehouse:** Nesse capítulo serão apresentados conceitos teóricos sobre *Data Warehousing*, assim como a maneira como foi desenvolvido e como funciona o ambiente de *Data Warehouse* para armazenamento de métricas de código fonte.

- **Capítulo 5 - Projeto de estudo de caso:** é apresentado o projeto do estudo de caso resultante do passo planejar Estudo de Caso, buscando demonstrar o escopo e elaborar um protocolo para o estudo de caso que será realizado. Elementos de pesquisa serão identificados e explicados como o problema a ser resolvido, os objetivos a serem alcançados no estudo de caso, quais os métodos de coleta e a análise dos dados.

- **Capítulo 6 - Conclusão:** Além das considerações finais dessa primeira parte do trabalho, serão descritos objetivos para o trabalho de conclusão de curso dois.

2 Métricas de Software

2.1 Processo de Medição

Segundo o (SOFTEX, 2013b) o processo Medição é um processo que apoia os processos de gerência e melhoria de processo e de produto, sendo um dos processos principais para gerenciar as atividades do ciclo de vida do trabalho e avaliar a viabilidade dos planos de trabalho. O propósito da Medição é coletar e analisar os dados relativos aos produtos desenvolvidos e aos processos implementados na organização e em seus trabalhos, de forma a apoiar o efetivo gerenciamento dos processos e demonstrar objetivamente a qualidade dos produtos(ISO/IEC 15939, 2008).

Ainda segundo o (SOFTEX, 2013b) Entende-se por método de medição uma sequência lógica de operações, descritas genericamente, usadas para quantificar um atributo com respeito a uma escala especificada. Esta escala pode ser nominal, ordinal ou de razão (de proporção), bem como definida em um intervalo.

- **Nominal:** A ordem não possui significado na interpretação dos valores (MEIRELLES, 2013)
- **Ordinal:** A ordem dos valores possui significado, porém a distância entre os valores não. (MEIRELLES, 2013)
- **Intervalo:** A ordem dos valores possui significado e a distância entre os valores também. Porém, a proporção entre os valores não necessariamente possui significado. (MEIRELLES, 2013)
- **Racional:** Semelhante a a medida com escala do tipo intervalo, porém a proporção possui significado. (MEIRELLES, 2013)

A ISO/IEC 15939 (2002) também divide o processo de medição em dois métodos diferentes, que se distinguem pela natureza do que é quantificado:

- **Subjetiva:** Quantificação envolvendo julgamento de um humano
- **Objetiva:** Quantificação baseada em regras numéricas. Essas regras podem ser implementadas por um humano.

2.2 Definição das métricas de software

As métricas de software são medidas resultantes da medição do produto ou do processo do *software* pelo qual é desenvolvido, sendo que o produto de *software* deve

ser visto como um objeto abstrato que se desenvolveu a partir de uma declaração inicial da necessidade de um sistema para um software finalizado, incluindo o código-fonte e as várias formas de documentação produzida durante o desenvolvimento Mills (1999). Estas medidas resultantes podem ser estudas para serem utilizadas para medir a produtividade e a qualidade do produto.

(MILLS, 1999) classifica as métricas de software como métricas de produtos ou métricas de processo, essa divisão pode ser vista na figura 3.

- **Métricas de produtos:** são as medidas do produto de software em qualquer fase do seu desenvolvimento, a partir dos requisitos do sistema. Métricas de produto podem medir a complexidade da arquitetura do software, o tamanho do programa(código-fonte), ou o número de páginas de documentos produzidos.

- **Métricas de processo:** são as medidas do processo de desenvolvimento de software, como o tempo de desenvolvimento global, o tipo de metodologia utilizada, ou o nível médio da experiência da equipe de programação.

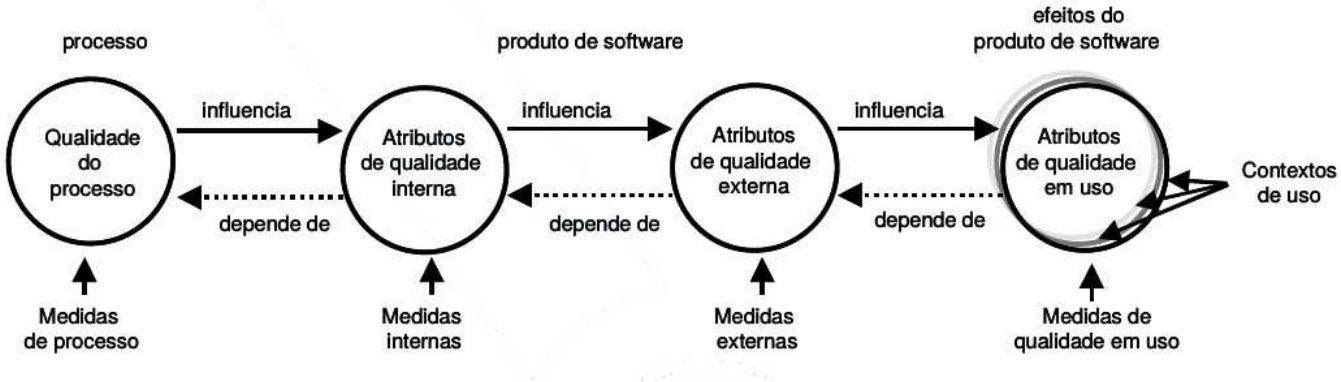

Figura 3 – Modelo de Qualidade do Produto da ISO/IEC 9126 (2001)

A figura 3 também pode ser notado a classificação das métricas de acordo com os diferentes tipos de medição, refletindo como as métricas influenciam nos contextos em que elas estão envolvidas, A qualidade do produto de software pode ser avaliada medindo-se os atributos internos (tipicamente medidas estáticas de produtos intermediários), os atributos externos (tipicamente pela medição do comportamento do código quando executado) ou os atributos de qualidade em uso (ISO/IEC 9126, 2001).

- **Métricas internas:** Aplicadas em um produto de software não executável, como código fonte. Oferecem aos usuários, desenvolvedores ou avaliadores o benefício de poder avaliar a qualidade do produto antes que ele seja executável.

- **Métricas externas:** Aplicadas a um produto de software executável, medindo o comportamento do sistema do qual o software é uma parte através de teste, operação ou mesmo obervação. Oferecem aos usuários, desenvolvedores ou avaliadores o

benefício de poder avaliar a qualidade do produto durante seu processo de teste ou operação.

- **Métricas de qualidade em uso:** Aplicadas para medir o quanto um produto atende as necessidades de um usuário para que sejam atingidas metas especificadas como eficácia, produtividade, segurança e satisfação.

2.3 Métricas de código fonte

A definição de código-fonte segundo a SCAM é qualquer descrição executável de um sistema de software sendo incluído código de máquina, linguagens de alto nível e por representações gráficas executáveis (HARMAN, 2010).

Segundo (MILLS, 1999) a maior parte do trabalho inicial em métricas de produto analisaram as características do código-fonte. A partir da experiência com métricas e modelos, tornou-se cada vez mais evidente que as informações de métricas obtidas anteriormente do ciclo de desenvolvimento pode ser de grande valor no controle do processo e dos resultados, o que sucedeu uma série de trabalhos tratando sobre o tamanho ou complexidade do software.

Neste capítulo será evidenciado em duas categorias as métricas de código-fonte que serão utilizadas neste trabalho de conclusão de curso, métricas de tamanho e complexidade e métricas de orientação a objetos. Estas métricas são objetivas e serão calculadas a partir da análise estática do código-fonte de um software.

2.3.1 Métricas de tamanho e complexidade

Cada produto do desenvolvimento de software é uma entidade física, como tal, pode ser descrito em termos de tamanho. Desde outros objetos físicos são facilmente mensuráveis (em comprimento, volume, massa, ou outra medida padrão), medir o tamanho do software deve ser relativamente simples e coerente de acordo com os princípios da teoria da medição. No entanto, na prática, a medição de tamanho apresenta grandes dificuldades(FENTON; PFLEEGER, 1998).

Segundo (HONGLEI; WEI; YANAN, 2009) as métricas de complexidade *software* pertencem as principais medições de software e é o principal método para assegurar a qualidade do *software*. Quanto menor a complexidade dos programas, melhor eles são, as métricas de complexidade também podem ser usadas para prever os defeitos ou erros. A seguir são apresentadas algumas métricas de tamanho e complexidade.

- **LOC** (*Lines of Code*): Métrica simples em que são contadas as linhas executáveis de um código, desconsiderando linhas em branco e comentários. (KAN, 2002)

- **ACCM** (*Average Cyclomatic Complexity per Method*): Mede a complexidade do programa, podendo ser representada através de um grafo de fluxo de controle. (MCCABE, 1976)

- **AMLOC** (*Average Method Lines of Code*): Indica a distribuição de código entre os métodos. Quanto maior o valor da métrica, mais pesado é o método. É preferível que haja muitos métodos com pequenas operações do que um método grande e de entendimento complexo. (MEIRELLES, 2013)

2.3.2 Métricas de Orientação a Objetos

Segundo (BUDD, 2002) a programação orientada a objetos tornou-se extremamente popular nos últimos anos, inúmeros livros e edições especiais de revistas acadêmicas e comerciais têm surgido sobre o assunto. A julgar por essa atividade frenética, programação orientada a objeto está sendo recebido com ainda mais entusiasmo do que vimos em ideias revolucionárias mais antigas, tais como programação estruturada "ou sistemas especialistas."

Há uma série de razões importantes pelas quais nas últimas duas décadas a programação orientada a objetos tornou-se o paradigma de programação dominante. A programação orientada a objeto possui uma ótima escala, desde o mais trivial dos problemas para a maioria das tarefas complexas. Ele fornece uma forma de abstração que reflete em técnicas de como pessoas usam para resolver problemas em sua vida cotidiana. E para a maioria da linguagens orientadas a objeto dominante há um número cada vez maior de bibliotecas que auxiliam no desenvolvimento de aplicações para muitos domínios (BUDD, 2002).

Serão adotadas neste trabalho as seguintes métricas de orientação a objetos:

- **ACC** (*Afferent Connections per Class* - Conexões Aferentes por Classe): Mede a conectividade entre as classes. Quanto maior a conectividade entre elas, maior o potencial de impacto que uma alteração pode gerar. (MEIRELLES, 2013)

- **ANPM** (*Average Number of Parameters per Method* - Média do Número de Parâmetros por Método): Indica a média de parâmetros que os métodos possuem. Um valor muito alto para quantidade de parâmetros pode indicar que o método está tendo mais de uma responsabilidade. (BASILI; ROMBACH, 1987)

- **CBO** (*Coupling Between Objects* - Acoplamento entre Objetos): Essa é uma métrica que diz respeito a quantas outras classes dependem de uma classe. É a conta das classes às quais uma classe está acoplada. Duas classes estão acopladas quando métodos de uma delas utilizam métodos ou variáveis de outra. Altos valores dessa métrica aumentam a complexidade e diminuem a manutenibilidade. (LAIRD, 2006).

- **DIT** (*Depth of Inheritance Tree* - Profundidade da Árvore de Herança): Responsável por medir quantas camadas de herança compõem uma determinada hierarquia de classes (LAIRD, 2006). Segundo Meirelles (2013), quanto maior o valor de DIT, maior o número de métodos e atributos herdados, portanto maior a complexidade.

- **LCOM4** (*Lack of Cohesion in Methods* - Falta de Coesão entre Métodos): A coesão de uma classe é indicada por quão próximas as variáveis locais estão relacionadas com variáveis de instância locais. Alta coesão indica uma boa subdivisão de classes. A LCOM mede a falta de coesão através dissimilaridade dos métodos de uma classe pelo emprego de variáveis de instância.(KAN, 2002). A métrica LCOM foi revista e passou a ser conhecida como LCOM4, sendo necessário para seu cálculo a construção de um gráfico não-orientado em que os nós são os atributos e métodos de uma classe. Para cada método deve haver uma aresta entre ele e outro método ou variável. O valor da LCOM4 é o número de componentes fracamente conectados a esse gráfico (MEIRELLES, 2013)

- **NOC** (*Number of Children* - Número de Filhos): É o número de sucessores imediatos, (portanto filhos) de uma classe. Segundo Laird (2006), altos valores indicam que a abstração da super classe foi diluída e uma reorganização da arquitetura deve ser considerada.

- **NOM** (*Number of Methods* - Número de Métodos): Indica a quantidade de métodos de uma classe, medindo seu tamanho. Classes com muitos métodos são mais difíceis de serem reutilizadas pois são propensas a serem menos coesas. (MEIRELLES, 2013)

- **NPA** (*Number of Public Attributes* - Número de Atributos Públicos): Mede o encapsulamento de uma classe, através da medição dos atributos públicos. O número ideal para essa métrica é zero (MEIRELLES, 2013)

- **RFC** (*Response For a Class* - Respostas para uma Classe): Kan (2002) define essa métrica como o número de métodos que podem ser executados em respostas a uma mensagem recebida por um objeto da classe.

2.4 Configurações de qualidade para métricas de código fonte

Meirelles (2013) apresentou uma abordagem para a observação das métricas de código-fonte em seu doutorado, onde estas métricas foram estudadas através de suas distribuições e associações. Foram avaliados a distribuição e correlações dos valores das métricas de 38 projetos de software livre, sendo coletado e analisado valores para cada métrica em mais de 344.872 classes e módulos. Segundo o próprio Meirelles (2013) entre as principais contribuições de sua tese foi a análise detalhada, em relação ao comportamento,

valores e estudos de caso, de 15 métricas de código-fonte.

Dentre os objetivos científicos da tese de Meirelles (2013) o mais crucial para este trabalho de conclusão de curso é o objetivo OC4 que trata das distribuições estatísticas dos valores de métricas em 38 projetos de software livre, a fim de compreender qual a abordagem estatística mais informativa para o monitoramento dessas métricas, bem como observar os valores frequentes para essas métricas, de forma a servirem de referência para projetos futuros.

(MEIRELLES, 2013) para conduzir o seu estudo qualitativo utilizou-se da técnica de estatística descritiva: percentil. O percentil separa os dados em cem grupos que apresentam o mesmo número de valores, sendo a definição do percentil de ordem px100(0<p<1), em um conjunto de dados de tamanho n, é o valor da variável que ocupa a posição px(n+1) do conjunto de dados ordenados. O percentil de ordem p (ou p-quantil) deixa px100% das observações abaixo dele na amostra ordenada. O Percentil 25 recebe o nome de primeiro quartil, o percentil 50 de segundo quartil ou mediana, e o percentil 75 de terceiro quartil. Uma das hipóteses que Meirelles (2013) utiliza é que só a partir do terceiro quartil será possível obter dados informativos sobre as métricas.

Após a aplicação da técnica percentil na série de dados das métricas de código-fonte obtidos da análise estática do código-fonte dos projetos de software livre, tabelas apresentando os valores percentis de cada métrica nos 38 projetos avaliados foram criadas, como a Tabela 1, que mostra os percentis para a métrica NOC.

	Mín	**1%**	**5%**	**10%**	**25%**	**50%**	**75%**	**90%**	**95%**	**99%**	**Máx**
Eclipse	0,0	0,0	0,0	0,0	0,0	0,0	0,0	2,0	3,0	10,0	243,0
Open JDK8	0,0	0,0	0,0	0,0	0,0	0,0	0,0	1,0	3,0	9,0	301,0
Ant	0,0	0,0	0,0	0,0	0,0	0,0	0,0	1,0	3,0	12,0	162,0
Checkstyle	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	2,0	5,0	144,0
Eclipse Metrics	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	2,0	11,0	24,0
Findbugs	0,0	0,0	0,0	0,0	0,0	0,0	0,0	1,0	2,0	5,0	55,0
GWT	0,0	0,0	0,0	0,0	0,0	0,0	0,0	39,0	1,0	8,0	398,0
Hudson	0,0	0,0	0,0	0,0	0,0	0,0	0,0	1,0	2,0	7,0	23,0
JBoss	0,0	0,0	0,0	0,0	0,0	0,0	0,0	1,0	2,0	6,0	256,0
Kalibro	0,0	0,0	0,0	0,0	0,0	0,0	0,0	1,0	2,0	9,0	101,0
Log4J	0,0	0,0	0,0	0,0	0,0	0,0	0,0	1,0	3,0	9,0	23,0
Netbeans	0,0	0,0	0,0	0,0	0,0	0,0	0,0	1,0	2,0	6,0	989,0
Spring	0,0	0,0	0,0	0,0	0,0	0,0	0,0	2,0	3,0	7,0	175,0
Tomcat	0,0	0,0	0,0	0,0	0,0	0,0	0,0	1,0	2,0	8,0	82,0

Tabela 1 – Percentis para métrica NOC em projetos Java extraídos de Meirelles (2013)

Através dos resultados obtidos para cada métrica, Meirelles (2013) observou que era possível identificar valores frequentes analisando os percentis. Na tabela 1, por exemplo, considerando o projeto **Open JDK8** como referência foram observados os intervalos

de valores de 0, como muito frequente, 1 e 2 como frequente, 3 como pouco frequente e acima de 3 como não frequente (MEIRELLES, 2013).

(RêGO, 2014) observando o trabalho de análise de percentis de (MEIRELLES, 2013) percebeu que é possível utilizar os intervalos de frequência obtidos como uma evidência empírica de qualidade do código-fonte. (RêGO, 2014) também renomeou os intervalos de frequência obtidos por (MEIRELLES, 2013), como na Tabela 2, a fim de facilitar a interpretação de métricas de código-fonte.

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

Tabela 2 – Nome dos Intervalos de Frequência extraídos de Rêgo (2014)

(RêGO, 2014) observando a diferença dos valores das métricas entre os projetos analisados e na tentativa de diminuir tamanha diferença, considerou dois cenários. Foi utilizado dois *softwares* como referências para cada uma das métricas na linguagem de programação Java. Em um primeiro cenário, foi analisado o *Open JDK8*, software que contia os menores valores percentis para as métricas. Já em um segundo cenário, foi considerado o Tomcat contendo os valores percentis mais altos. A tabela 3 mostra o resultado desta análise:

Métrica	Intervalo Qualitativo	OpenJDK8 Metrics	Tomcat Metrics
LOC	Excelente	[de 0 a 33]	[de 0 a 33]
	Bom	[de 34 a 87]	[de 34 a 105]
	Regular	[de 88 a 200]	[de 106 a 276]
	Ruim	[acima de 200]	[acima de 276]
ACCM	Excelente	[de 0 a 2,8]	[de 0 a 3]
	Bom	[de 2,9 a 4,4]	[de 3,1 a 4,0]
	Regular	[de 4,5 a 6,0]	[de 4,1 a 6,0]
	Ruim	[acima de 6]	[acima de 6]
AMLOC	Excelente	[de 0 a 8,3]	[de 0 a 8]
	Bom	[de 8,4 a 18]	[de 8,1 a 16,0]
	Regular	[de 19 a 34]	[de 16,1 a 27]
	Ruim	[acima de 34]	[acima de 27]
ACC	Excelente	[de 0 a 1]	[de 0 a 1,0]
	Bom	[de 1,1 a 5]	[de 1,1 a 5,0]
	Regular	[de 5,1 a 12]	[de 5,1 a 13]
	Ruim	[acima de 12]	[acima de 13]
ANPM	Excelente	[de 0 a 1,5]	[de 0 a 2,0]
	Bom	[de 1,6 a 2,3]	[de 2,1 a 3,0]
	Regular	[de 2,4 a 3,0]	[de 3,1 a 5,0]
	Ruim	[acima de 3]	[acima de 5]
CBO	Excelente	[de 0 a 3]	[de 0 a 2]
	Bom	[de 4 a 6]	[de 3 a 5]
	Regular	[de 7 a 9]	[de 5 a 7]
	Ruim	[acima de 9]	[acima de 7]
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]
LCOM4	Excelente	[de 0 a 3]	[de 0 a 3]
	Bom	[de 4 a 7]	[de 4 a 7]
	Regular	[de 8 a 12]	[de 8 a 11]
	Ruim	[acima de 12]	[acima de 11]
NOC	Excelente	[0]	[1]
	Bom	[1 a 2]	[1 a 2]
	Regular	[3]	[3]
	Ruim	[acima de 3]	[acima de 3]
NOM	Excelente	[de 0 a 8]	[de 0 a 10]
	Bom	[de 9 a 17]	[de 11 a 21]
	Regular	[de 18 a 27]	[de 22 a 35]
	Ruim	[acima de 27]	[acima de 35]
NPA	Excelente	[0]	[0]
	Bom	[1]	[1]
	Regular	[de 2 a 3]	[de 2 a 3]
	Ruim	[acima de 3]	[acima de 3]
RFC	Excelente	[de 0 a 9]	[de 0 a 11]
	Bom	[de 10 a 26]	[de 12 a 30]
	Regular	[de 27 a 59]	[de 31 a 74]
	Ruim	[acima de 59]	[acima de 74]

Tabela 3 – Configurações para os Intervalos das Métricas para Java extraídas de Rêgo (2014)

2.5 Cenários de limpeza

(MACHINI et al., 2010) apresenta uma maneira de interpretar os valores das métricas através de cenários problemáticos e suas possíveis melhorias, para que as métricas possam ser mais facilmente incorporadas no cotidiano dos programadores. Machini et al. (2010) também apresenta um estilo de programação baseado no paradigma da Orientação a Objetos que busca o que denominamos de "Código Limpo", concebido e aplicado por renomados desenvolvedores de software como Robert C. Martin (MARTIN, 2008) e Kent Beck (BECK, 2007)

Segundo (BECK, 2007) um código limpo está inserido em um estilo de programação que busca a proximidade a três valores: expressividade, simplicidade e flexibilidade.

- **Expressividade:** Um código se expressa bem quando alguém que o lê é capaz de compreendê-lo e modificá-lo. Beck (2007) destaca que quando foi necessário modificar um código, ele gastou muito mais tempo lendo o que já havia sido feito do que escrevendo sua modificação.

- **Simplicidade:** Um código é expressivo quando pode ser facilmente lido nas diferentes camadas de abstração, havendo uma redução da quantidade de informação que o leitor deve compreender para fazer alterações. Eliminar o excesso de complexidade faz com que aqueles que estejam lendo o código o entenda mais rapidamente.

- **Flexibilidade:** Capacidade de estender a aplicação alterando o mínimo possível a estrutura já criada.

(MACHINI et al., 2010) apresenta em seu trabalho diversas técnicas para a obtenção de um código limpo, um resumo destas técnicas são apresentadas nas tabelas 4, 5, 6 e 7.

Técnicas de Limpeza	Descrição	Contribuições	Consequências
Composição de Métodos	Compor os métodos em chamadas para outros rigorosamente no mesmo nível de abstração abaixo.	• Facilidade de entendimento de métodos menores • Criação de métodos menores com nomes explicativos	• Menos Operações por Método • Mais Parâmetros de Classe • Mais Métodos na Classe
Métodos Explicativos	Criar um método que encapsule uma operação pouco clara geralmente associada a um comentário	• O código cliente do método novo terá uma operação com nome que melhor se encaixa no contexto.	• Mais métodos na classe.
Métodos como Condicionais	Criar um método que encapsule uma expressão booleana para obter condicionais mais claros.	• Facilidade na leitura de condicionais no código cliente. • Encapsulamento de uma expressão booleana.	• Mais métodos na classe.
Evitar Estruturas Encadeadas	Utilizar a composição de métodos para minimizar a quantidade de estruturas encadeadas em cada método (if, else).	• Facilidade para a criação de testes. • Cada método terá estruturas mais simples e fáceis de serem compreendidas.	• Menos Estruturas encadeadas por método (if e else) • Benefícios do Uso de Composição de Métodos

Tabela 4 – Conceitos de Limpeza levantados por Machini et al. (2010) e adaptados por Rêgo (2014) parte 1.

Técnicas de Limpeza	Descrição	Contribuições	Consequências
Cláusulas Guarda	Criar um retorno logo no inicio de um método ao invés da criação de estruturas encadeadas com if sem else.	• Estruturas de condicionais mais simples. • Leitor não espera por uma contrapartida do condicional(ex:if sem else).	• Menos estruturas encadeadas na classe.
Objeto Método	Criar uma classe que encapsule uma operação complexa simplificando a original(cliente).	• O código cliente terá um método bastante simples. • Nova classe poderá ser refatorada sem preocupações com alterações no código cliente. • Nova classe poderá ter testes separados.	• Menos operações no método cliente. • Menos responsabilidades da classe cliente. • Mais classes. • Mais acoplamento da classe cliente com a nova classe.
Evitar *Flags* como Argumentos	Ao invés de criar um método que recebe uma flag e tem diversos comportamentos, criar um método para cada comportamento.	• Leitor não precisará entender um método com muitos condicionais. • Testes de unidade independentes.	• Mais métodos na classe.

Tabela 5 – Conceitos de Limpeza levantados por Machini et al. (2010) e adaptados por Rêgo (2014) parte 2.

Técnicas de Limpeza	Descrição	Contribuições	Consequências
Maximizar a Coesão	Quebrar uma classe que não segue o Princípio da Responsabilidade Única: as classes devem ter uma única responsabilidade, ou seja, ter uma única razão para mudar.	• Cada classe terá uma única responsabilidade. • Cada classe terá seus testes independentes. • Sem interferências na implementação das responsabilidades.	• Mais Classes • Menos Métodos em cada Classe • Menos Atributos em cada Classe
Objeto como Parâmetro	Localizar parâmetros que formam uma unidade e criar uma classe que os encapsule.	• Menor número de parâmetros facilita testes e legibilidade. • Criação de uma classe que poderá ser reutilizada em outras partes do sistema.	• Menos Parâmetros sendo passados para Métodos • Mais Classes
Parâmetros como Variável de Instância	Localizar parâmetro muito utilizado pelos métodos de uma classe e transformá-lo em variável de instância.	• Não haverá a necessidade de passar longas listas de parâmetro através de todos os métodos.	• Menos Parâmetros passados pela Classe • Possível diminuição na coesão
Uso de Exceções	Criar um fluxo normal separado do fluxo de tratamento de erros utilizando exceções ao invés de valores de retornos e condicionais.	• Clareza do fluxo normal sem tratamento de erros através de valores de retornos e condicionais.	• Menos Estruturas encadeadas.

Tabela 6 – Conceitos de Limpeza levantados por Machini et al. (2010) e adaptados por Rêgo (2014) parte 3.

Técnicas de Limpeza	Descrição	Contribuições	Consequências
Delegação de Tarefa	Transferir um método que utiliza dados de uma classe "B"para a "B".	• Redução do acoplamento entre classes. • Proximidade dos métodos e dados sobre os quais trabalham.	• Menos métodos na classe inicial. • Mais métodos na classe que recebe o novo método.
Objeto Centralizador	Criar uma classe que encapsule uma operação com alta dependência entre classes.	• Simplificação da classe cliente. • Redução do acoplamento da classe cliente com as demais. • Nova classe poderá receber testes e melhorias independentes.	• Menos operações no método cliente. • Menos responsabilidades da classe cliente. • Mais classes. • Mais acoplamento da classe cliente com a nova classe.
Uso Excessivo de Herança	Localizar uso excessivo de herança e transformá-lo em agregação simples.	• Redução do do acoplamento entre as classes.	• Maior Flexibilidade de Adição de Novas Classes. • Menor Acoplamento entre as classes.
Exposição Pública Excessiva	Localizar uso excessivo de parâmetros públicos e transformá-lo em parâmetros privados.	• Menor Acoplamento entre as classes.	• Maior Encapsulamento de Parâmetros.

Tabela 7 – Conceitos de Limpeza levantados por Machini et al. (2010) e adaptados por Rêgo (2014) parte 4.

Após apresentar as técnicas para obtenção de um código limpo Machini et al. (2010)adota uma abordagem baseada em cenários para identificar trechos de código com características indesejáveis, onde estes cenários devem possuir um contexto criado a partir

de poucos conceitos de código limpo no qual um pequeno conjunto de métricas é analisado e interpretado através da combinação de seus valores. A ideia principal desta abordagem é facilitar melhorias de implementação e a procura por problemas quanto a limpeza de código através da aproximação dos valores das métricas com os esperados nos contextos de interpretação (MACHINI et al., 2010).

(RêGO, 2014) utiliza a mesma abordagem de cenários que Machini et al. (2010) onde algumas técnicas de limpeza de código apresentadas nas tabelas 4, 5, 6 e 7 são correlacionadas com as métricas da Seção 2.3 e utiliza a configuração do *Open JDK8* considerando como valores altos os valores obtidos pelos intervalos Regular e Ruim tal como mostrados na Tabela 3.

Aproveitando inicialmente os cenários **Classe pouco coesa** e **Interface dos métodos** extraídos de Machini et al. (2010), Rêgo (2014) elaborou mais alguns cenários de limpeza. O resultado dessa atividade pode ser visto na tabela 8:

Cenário de Limpeza	Conceito de Limpeza	Características	Recomendações	Forma de Detecção pelas Métricas de Código-Fonte	Padrões de Projeto Associados
Classe Pouco Coesa	Maximização da Coesão	Classe Subdivida em grupos de métodos que não se relacionam	Reduzir a subdvisão da Classe	Intervalos Regulares e Ruins de LCOM4, RFC.	*Chain of Reponsabilities, Mediator, Decorator.*
Interface dos Métodos	Objetos como Parâmetro e Parâmetro como Variáveis de Instância	Elevada Média de parâmetros repassados pela Classe	Minimizar o número de Parâmetros.	Intervalos Regulares e Ruins de ANPM.	*Facade, Template Method, Strategy, Command, Mediator, Bridge.*
Classes com muitos filhos	Evitar Uso Excessivo de Herança	Muitas Sobreescritas de Métodos	Trocar a Herança por uma Agregação.	Intervalos Regulares e Ruins de NOC.	*Composite, Prototype, Decorator, Adapter.*
Classe com métodos grandes e/ou muitos condicionais	Composição de Métodos, Evitar Estrutura Encadeadas Complexas	Grande Número Efetivo de Linhas de Código	Reduzir LOC da Classe e de seus métodos, Reduzir a Complexidade Ciclomática e Quebrar os métodos.	Intervalos Regulares e Ruins de AMLOC, ACCM.	*Chain of Reponsabilities, Mediator, Flyweight* .
Classe com muita Exposição	Parâmetros Privados	Grande Número de Parâmetros Públicos	Reduzir o Número de Parâmetros Públicos.	Intervalos Regulares e Ruins de NPA.	*Facade, Singleton.*
Complexidade Estrutural	Maximização da Coesão	Grande Acoplamento entre Objetos	Reduzir a a quantidade de responsabilidades dos Métodos.	Intervalos Regulares e Ruins de CBO e LCOM4.	*Chain of Responsabilities, Mediator, Strategy.*

Tabela 8 – Cenários de Limpeza extraídos de Rêgo (2014)

2.6 Considerações Finais do Capítulo

Esse capítulo apresentou a fundamentação teórica sobre aquilo que é medido e monitorado pela solução proposta. Além de uma definição sobre o que são métricas de código fonte e quais são seus intervalos qualitativos, também foi apresentada nesse capítulo a maneira como elas foram relacionadas a cenários de limpeza. O próximo capítulo será responsável por apresentar a teoria acerca de *Data Warehousing* e como seu uso pode servir no monitoramento das métricas apresentadas nesse capítulo.

3 Contratações de Fornecedores de Desenvolvimento de Software

3.1 Importância da Contratação de Fornecedores de Desenvolvimento de Software

3.2 Normas, Processos e Legislação Pertinentes à Contratação

3.3 Contrato do Órgão X

4 *Data Warehouse*

4.1 Ciclo de vida de um *Data Warehousing*

4.1.1 *Extraction-Transformation-Load*

4.2 Modelagem Dimensional

4.2.1 OLAP *On-Line Analytic Processing*

5 Projeto de estudo de Caso

Este capítulo irá apresentar o projeto do estudo de caso que foi realizado no passo planejar estudo de caso, este passo consiste na elaboração de um protocolo para o estudo de caso, na identificação do problema e na definição das questões e objetivos de pesquisa. O método para coleta dos dados e como eles serão analisados também serão expostos neste capítulo.

5.1 Definição

Segundo (YIN, 2001) o estudo de caso é um conjunto de procedimentos pré-especificados para se obter uma investigação empírica que investiga um fenômeno contemporâneo dentro de seu contexto da vida real, especialmente quando os limites entre o fenômeno e o contexto não estão claramente definidos. Uma grande vantagem do estudo de caso é a sua capacidade de lidar com uma ampla variedade de evidências - documentos, artefatos, entrevistas e observações - além do que pode estar disponível no estudo histórico convencional. Além disso, em algumas situações, como na observação participante, pode ocorrer manipulação informal.

Buscando maior entendimento a respeito do estudo de caso proposto, foram criadas algumas perguntas que são fundamentais para o seu entendimento:

- Qual o escopo do estudo de caso?
- Qual o problema a ser tratado?
- Qual a questão de pesquisa relacionada a esse problema?
- Quais são os objetivos a serem alcançados nessa pesquisa?
- Como foi a seleção do estudo de caso?
- Qual é fonte dos dados coletados nessa pesquisa e qual o método de coleta?

Com o objetivo da elucidação do escopo do estudo de caso proposto neste trabalho apresentado na Figura 4,foi apresentado nos capítulos antecedentes, um estudo teórico relacionado à métricas de software, contratação de serviços de TI em administrações públicas brasileiras e de Data *Warehouse*. Também foi apresentado uma solução para o monitoramento de Métricas de Código-Fonte com suporte de um ambiente de Data *Warehousing*.

Portanto o fundamento da proposta deste trabalho está ligado à análise e coleta de

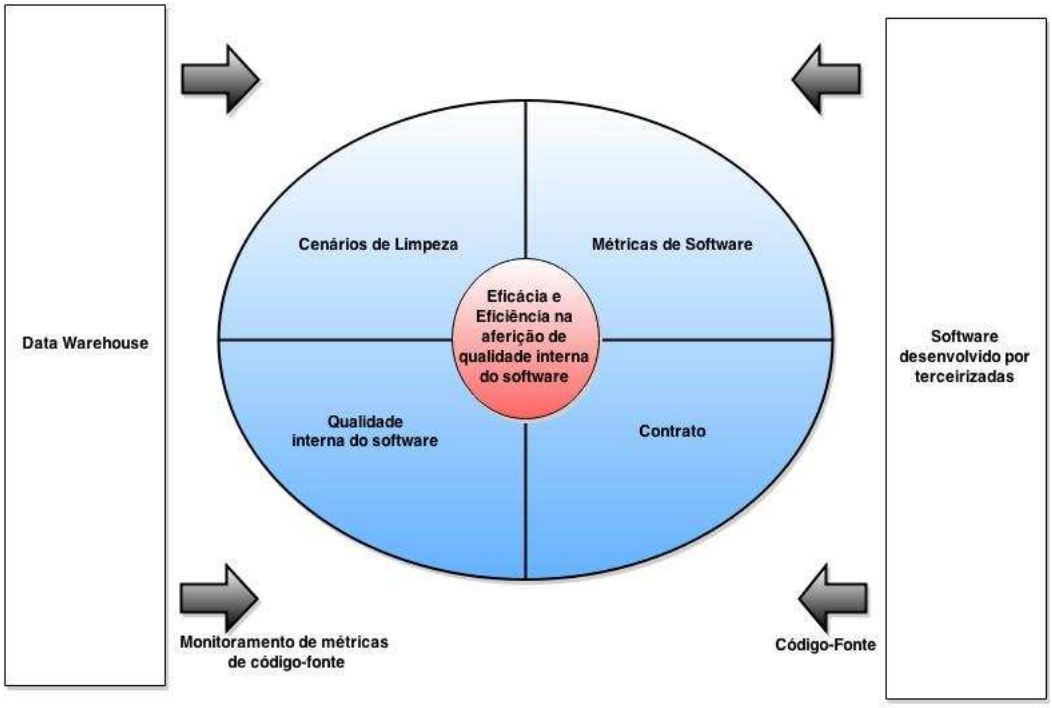

Figura 4 – Escopo do Estudo de Caso

métricas de código-fonte e de cenários de limpeza de um *software*, desenvolvido por uma empresa terceirizada contratada por um órgão público brasileiro, utilizando uma solução de DW. Onde o foco será será evidenciar a eficácia e a eficiência da proposta por Rêgo (2014) no aferimento de qualidade do software adquirido de uma empresa terceirizada pela equipe responsável do Órgão público e seus demais envolvidos.

Para as demais perguntas serem respondidas, foi estruturado neste trabalho um problema, uma questão de pesquisa, objetivos e questões específicas conforme ilustrado na Figura 5.

5.1.1 Problema

O problema refere-se ao problema de pesquisa identificado na seção "Problema"do capítulo um.

A falta de capacidade da administração pública de aferir a qualidade interna dos produtos de software adquiridos e desenvolvidos por empresas contratadas.

5.1.2 Questão de Pesquisa

A questão de pesquisa refere-se a questão de pesquisa identificada na seção "Questão de Pesquisa"do capítulo um.

O uso de DW para o monitoramento de métricas de código fonte para

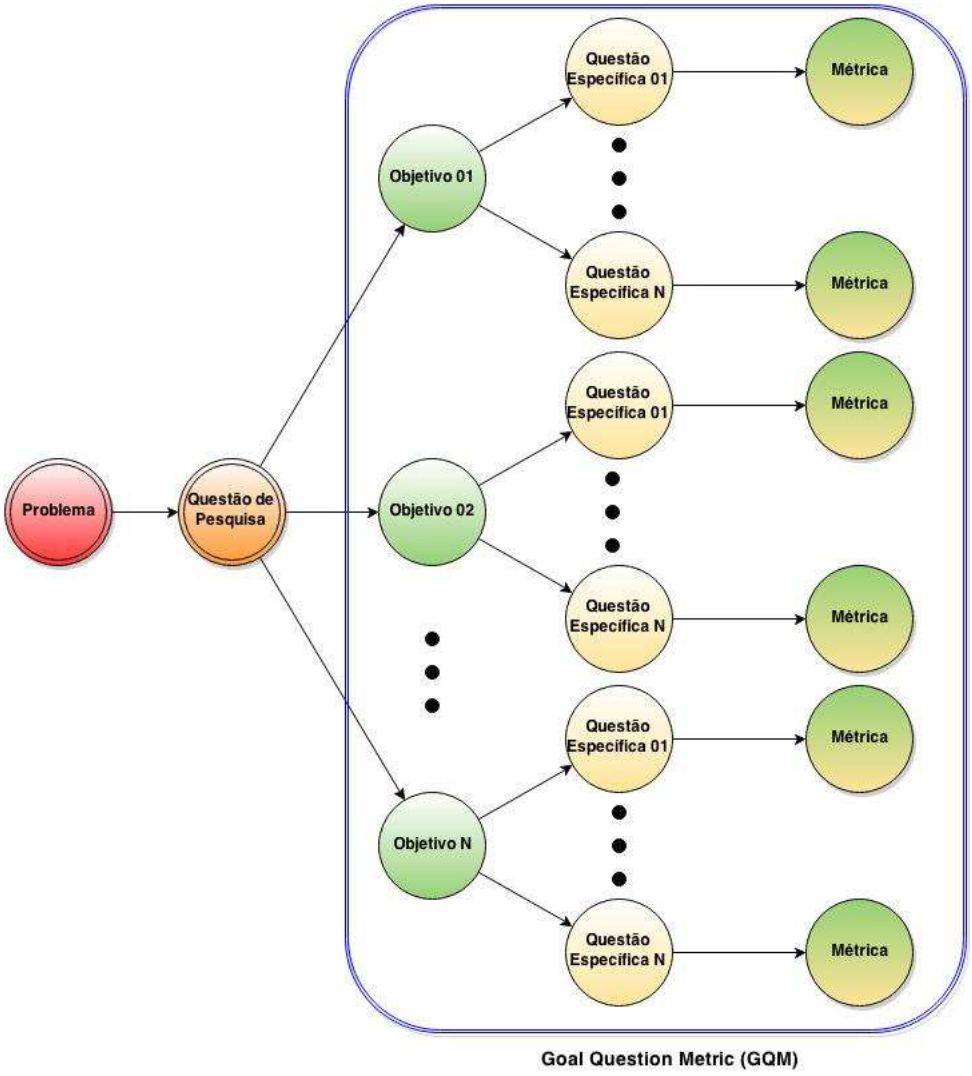

Figura 5 – Estrutura do Estudo de Caso

assistir ao processo de aferição de qualidade interna dos produtos de software desenvolvido por terceirizadas, do ponto de vista da equipe de qualidade no desenvolvimento de software no órgão público, é eficaz e eficiente?

Para atender a questão de pesquisa foi utilizado o mecanismo *goal-question-metrics*. Segundo (BASILI; CALDIERA; ROMBACH, 1996) o resultado da aplicação do GQM é a especificação de um sistema de medida destinada a um conjunto especifico de questões e um conjunto de regras para a interpretação dos dados da medição. O GQM é uma estrutura hierárquica que começa com um objetivo que especifica o propósito da medição, objeto a ser medido, questão a ser medida e o ponto de vista de que a medida seja tomada. O objetivo é refinado em várias questões, cada questão é transformada em métrica(objetiva ou subjetiva)(BASILI; CALDIERA; ROMBACH, 1996). A estrutura do GQM está a seguir.

Objetivo 01: Avaliar a eficácia e eficiência do uso de *Data Warehouse* para mo-

nitoramento de métricas de código fonte.

QE01: Quantas tomadas de decisão foram realizadas pela equipe baseando-se no uso da solução desenvolvida em um <período de tempo>?

Fonte: Registro de observação em campo.

Métrica: Número de decisões tomadas/tempo.

QE02: Quantas tomadas de decisão ao todo foram realizadas pela equipe baseando-se no uso da solução desenvolvida?

Fonte: Registro de observação em campo.

Métrica: Número de decisões tomadas.

QE03: Qual a avaliação da equipe de qualidade quanto a detecção de cenários de limpeza de código?

Fonte: Questionário com equipe de qualidade.

Métrica: muito bom, bom, regular, ruim, muito ruim.

QE04: Com que frequência a equipe de qualidade encontra falhas relacionados à utilização da ferramenta em um determinado intervalo de tempo?

Fonte: Registro de observação em campo.

Métrica: Quantidade de falha / tempo (release, sprint).

Interpretação da métrica: Quanto mais próximo de zero melhor 0=<X

QE05: Qual a quantidade total de falhas encontradas pela equipe de qualidade relacionadas à utilização da ferramenta?

Fonte: Registro de observação em campo (áudio/vídeo).

Métrica: Quantidade de falhas.

QE06: Qual a proporção do uso da ferramenta para tomada de decisões?

Fonte: Questionário com equipe de qualidade.

Métrica: Número de decisões tomadas / número de vezes que a solução foi usada.

QE07: Qual a quantidade de cenários que foram corrigidos após utilização da

solução?

Fonte: Código fonte.

Métrica: Números de cenários corrigidos / número de cenários encontrados.

QE08: Qual o nível de satisfação do uso da solução em comparação à solução anterior?

Fonte: Equipe de qualidade.

Métrica: Muito satisfeito, Satisfeito, Neutro, Insatisfeito, Muito Insatisfeito.

QE09: Qual a taxa de oportunidade de melhoria de código em uma intervalo de tempo (sprint, release)?

Fonte: Código fonte.

Métrica: Taxa de oportunidade de melhoria de código:

$$T_r = \frac{\sum_{i=1}^{n} Ce_i}{\sum_{i=1}^{n} Cl_i}$$

Ce é o total de cenários de limpezas identificados e Cl é o total de classes em um intervalo de tempo (sprint, release).

Interpretação da métrica:

- Número de classes crescente e constante, Número de oportunidade de melhoria estável: Projeto cresceu mais dos que o cenários, cenários podem ou não estar sendo tratados.
- Número de classes crescente e constante, Número de oportunidade de melhoria crescendo: Projeto cresce junto com cenários que não são tratados com eficiência ou não são tratados
- Número de classes crescente ou não, mas constante, número de oportunidade de melhoria diminuindo: Projeto cresce e cenários são tratados com eficiência.

5.2 *Background*

Referente ao contexto deste trabalho, o principal *background* é o trabalho desenvolvido por Rêgo (2014), no qual utilizaremos sua solução para monitorar as métricas de código-fonte utilizando *Data Warehouse*. Novello (2006) utilizou *Data Warehouse* para monitoramento de métricas no processo de desenvolvimento de software, porém são métricas voltadas para a qualidade do processo de desenvolvimento de software mas são diretamente relacionadas ao código-fonte. Portanto o levantamento bibliográfico realizado, não

foram encontrados estudos que utilizassem *Data Warehouse* para aferir qualidade interna do *software* aplicando coleta de métricas de código-fonte e geração de cenários de limpeza assim como foi feito por Rêgo (2014).

5.3 Seleção

Os dados foram coletados na CAIXA porque ??????????????????????????????

5.4 Fonte dos dados coletados e método de coleta

Os dados deste estudo de caso,serão coletados através de registros de observações em campo, questionários, entrevistas e resultados gerados pela própria solução utilizando o código-fonte de um ou mais sistemas desenvolvidos por empresas contratadas pelo órgão público.

Os registros oriundos de observações em campo são gerados a partir da equipe responsável pela tomada de decisões de qualidade e são coletados durante as visitas realizadas no órgão público. Nessas visitas, a solução proposta é utilizada e qualquer atitude relacionada ao seu uso pela equipe é registrada e será do ponto de vista qualitativo e quantitativo.

A adoção de questionários também foi utilizada tanto para dados qualitativos quanto para dados quantitativos á respeito da solução de DW vista pela equipe responsável e os demais envolvidos na qualidade de *software* no órgão público, no que diz respeito à tomadas de decisões, detecção de cenários de limpeza de código, falhas relacionados à utilização da ferramenta adotada na solução de DW, ao nível de satisfação no uso da solução e as taxas de oportunidades de melhoria de código.

Os gráficos e resultados gerados pela solução de DW, serão coletados a medida que a solução for utilizada pela equipe de qualidade ao longo das releases dos projetos adotados.

5.5 Ameaças a validade do estudo de caso

Yin (2001) descreve como principais ameaças relacionadas à validade do estudo de caso as ameaças relacionadas à validade do constructo, à validade interna, à validade externa e à confiabilidade. As quatro ameaças definidas por ele, bem como a forma usada nesse trabalho para preveni-las, são descritas da seguinte maneira:

- **Validade do Constructo:** A validade de construção está presente na fase de coleta de dados, quando deve ser evidenciado as múltiplas fontes de evidência e a coleta de um conjunto de métricas para que se possa saber exatamente o que medir e quais

dados são relevantes para o estudo, de forma a responder as questões de pesquisa (YIN, 2001). Buscou-se garantir a validade de construção ao definir objetivos com evidências diferentes. Estas, por sua vez, estão diretamente relacionadas com os objetivos do estudo de caso e os objetivos do trabalho.

- **Validade interna:** Para Yin (2001) o uso de várias fontes de dados e métodos de coleta permite a triangulação, uma técnica para confirmar se os resultados de diversas fontes e de diversos métodos convergem. Dessa forma é possível aumentar a validade interna do estudo e aumentar a força das conclusões. A triangulação de dados se deu pelo resultado da solução de *Data Warehouse* que utiliza o código-fonte (explicada no capítulo **??**), pela análise de questionários e pelos dados coletados através de entrevistas.

- **Validade externa:** Por este ser um caso único, a generalização do estudo de caso se dá de maneira pobre (YIN, 2001). Assim é necessária a utilização do estudo em múltiplos casos para que se comprove a generalidade dos resultados. Como este trabalho é o primeiro a verificar a eficácia e eficiência da solução para o estudo de caso no órgão, não há como correlacionar os resultados obtidos a nenhum outro estudo.

- **Confiabilidade:** Com relação a confiabilidade, Yin (2001) associa à repetibilidade, desde que seja usada a mesma fonte de dados. Nesse trabalho o protocolo de estudo de caso apresentado nessa seção garantem a repetibilidade desse trabalho e consequentemente a validade relacionada à confiabilidade.

5.6 Processo de análise dos dados

Análise dos dados coletados durante o estudo de caso a ser realizado no TCU? será feita através de 4 etapas:

- **Categorização:** Organização dos dados em duas categorias - qualitativos e quantitativos. Os dados qualitativos referem-se aos questionários realizados. Os dados quantitativos, por sua vez, referem-se aos valores numéricos da solução de DW para monitoramento de métricas.

- **Exibição:** Consiste na organização dos dados coletados para serem exibidos através de gráficos, tabelas e texto para poderem ser analisados.

- **Verificação:** Atestar padrões, tendências e aspectos específicos dos significados dos dados. Procurando assim gerar uma discussão e interpretação de cada dado exibido.

- **Conclusão:** Agrupamento dos resultados mais relevantes das discussões e interpretações dos dados anteriormente apresentados.

5.7 Considerações finais do capítulo

Esse capítulo teve como objetivo apresentar o protocolo de estudo de caso que será adotado na continuação deste trabalho. No trabalho de conclusão de curso dois serão detalhados o órgão público onde será aplicado o projeto de estudo de caso desenhado neste trabalho, os termos do contrato do órgão e a empresa contratada referentes a qualidade de *software* e serão apresentados os resultados obtidos para as questões específicas apresentadas neste capítulo assim como a interpretação dos mesmos possibilitando responder a questão de pesquisa abordada.

6 Conclusão

Referências

BASILI, V. R.; CALDIERA, G.; ROMBACH, H. D. *The Goal Question Metric Approach*. [S.l.]: Encyclopedia of Software Engineering, 1996. Citado na página 41.

BASILI, V. R.; ROMBACH, H. D. *TAME: Integrating Measurement into Software Environments*. 1987. Disponível em: <http://drum.lib.umd.edu//handle/1903/7517>. Citado na página 24.

BECK, K. *Extreme Programming Explained*. [S.l.]: Addison Wesley, 1999. Citado na página 15.

BECK, K. *Implementation Patterns*. 1. ed. [S.l.]: Addison-Wesley Professional, 2007. Citado na página 29.

BRASIL. *Acórdão-381/2011-TCU-Plenário*. [S.l.], 2011. Disponível em: <https://contas.tcu.gov.br/juris/SvlHighLight?key=ACORDAO-LEGADO-89657&texto=2b4e554d41434f5244414f2533413338312b414e442b2b4e55RELACAO-LEGADO;DECISAO-LEGADO;SIDOC;ACORDAO-RELACAO-LEGADO;>. Citado na página 16.

BRASIL. Instrução normativa número 4. 2014. Citado na página 15.

BUDD, T. *An introduction to object-oriented programming*. 3rd ed. ed. Boston: Addison-Wesley, 2002. ISBN 0201760312. Citado na página 24.

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 23.

FOWLER, M. *Refactoring: improving the design of existing code*. [S.l.]: Addison-Wesley Professional, 1999. Citado na página 15.

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. Citado na página 23.

HONGLEI, T.; WEI, S.; YANAN, Z. The research on software metrics and software complexity metrics. In: . IEEE, 2009. ISBN 978-1-4244-5422-8, 978-0-7695-3930-0. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5385114>. Citado na página 23.

ISO/IEC 15939. *ISO/IEC 15939: Software Engineering - Software Measurement Process*. [S.l.], 2002. Citado 2 vezes nas páginas 15 e 21.

ISO/IEC 15939. *ISO/IEC 12207: System and Software Engineering - Software Life Cycle Processes*. [S.l.], 2008. Citado na página 21.

ISO/IEC 9126. *ISO/IEC 9126-1: Software Engineering - Product Quality*. [S.l.], 2001. Citado 2 vezes nas páginas 9 e 22.

KAN, S. H. *Metrics and models in software quality engineering*. [S.l.]: Addison Wesley, 2002. Citado 2 vezes nas páginas 23 e 25.

LAIRD, M. C. B. L. M. *Software measurement and estimation: A practical approach*. [S.l.]: Wiley-IEEE Computer Society Press, 2006. Citado 2 vezes nas páginas 24 e 25.

LEITE, J. C. Terceirização em informática sob a ótica do prestador de serviços. *Revista de Administração de Empresas*, v. 37, n. 4, 1997. Disponível em: <http://www.scielo.br/pdf/rae/v37n4/a08v37n4>. Citado na página 15.

MACHINI, J. a. et al. *Código Limpo e seu Mapeamento para Métricas de Código-Fonte*. [S.l.]: Universidade de São Paulo, 2010. Citado 7 vezes nas páginas 10, 29, 30, 31, 32, 33 e 34.

MARTIN, R. C. *Clean Code: A Handbook of Agile Software Craftsmanship*. [s.n.], 2008. 464 p. ISBN 9780132350884. Disponível em: <http://portal.acm.org/citation.cfm?id=1388398>. Citado na página 29.

MCCABE, T. J. A Complexity Measure. *IEEE Transactions Software Engineering*, v. 2, n. 4, p. 308–320, December 1976. Citado na página 24.

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 6 vezes nas páginas 10, 21, 24, 25, 26 e 27.

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, abr. 1999. ISSN 1022-7091. Citado 2 vezes nas páginas 22 e 23.

NOVELLO, T. C. *Uma abordagem de Data Warehouse para análise de processos de desenvolvimento de software*. phdthesis — Pontifícia Universidade Católica do Rio Grande do Sul, 2006. Disponível em: <http://tardis.pucrs.br/dspace/handle/10923/1570>. Citado na página 43.

RêGO, G. B. Monitoramento de métricas de código-fonte com suporte de um ambiente de data warehousing: um estudo de caso em uma autarquia da administração pública federal. 2014. Disponível em: <http://bdm.unb.br/handle/10483/8069>. Citado 14 vezes nas páginas 10, 15, 16, 27, 28, 30, 31, 32, 33, 34, 35, 40, 43 e 44.

SOFTEX. *MPS. BR-Guia de Aquisição*. [S.l.], 2013. Disponível em: <http://www.softex.br/wp-content/uploads/2013/07/MPS.BR_Guia_de_Implementacao_SV_Parte_2_20132.pdf>. Citado na página 15.

SOFTEX. MPS BR-guia de implementação – parte 2: Fundamentação para implementação do nível f do mr-mps-sv:2012. 2013. Citado na página 21.

WILLCOCKS, L.; LACITY, M. *Global information technology outsourcing*. [S.l.: s.n.], 2001. Citado na página 15.

WOHLIN, C. et al. *Experimentation in software engineering*. [S.l.]: Springer, 2012. Citado na página 18.

YIN, R. *Estudo de caso: planejamento e métodos*. [S.l.]: Bookman, 2001. Citado 4 vezes nas páginas 17, 39, 44 e 45.

APÊNDICE A – Descrição do Processo de ETL no Kettle