A UTILIDADE DOS VALORES REFERÊNCIA DE MÉTRICAS NA AVALIAÇÃO DA QUALIDADE DE SOFTWARES ORIENTADOS POR OBJETO.

PRISCILA PEREIRA DE. SOUZA

A UTILIDADE DOS VALORES REFERÊNCIA DE MÉTRICAS NA AVALIAÇÃO DA QUALIDADE DE SOFTWARES ORIENTADOS POR OBJETO.

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais – Departamento de Ciência da Computação. como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Orientadora: Mariza Andrade da Silva Bigonha
Coorientadora: Kecia Aline Marques Ferreira

Belo Horizonte

Outubro de 2016

Souza, Priscila Pereira de.

P863o A Utilidade dos Valores Referência de Métricas na Avaliação da Qualidade de Softwares Orientados por Objeto. / Priscila Pereira de. Souza. — Belo Horizonte, 2016
xxiii, 116 f. : il. ; 29cm

Dissertação (mestrado) — Universidade Federal de Minas Gerais – Departamento de Ciência da Computação.

Orientador: Mariza Andrade da Silva Bigonha
Coorientadora: Kecia Aline Marques Ferreira.

Métricas de Software, Qualidade de Software, Valores Referência, Predição de Falhas, *Bad Smell*.I. Orientador. II. Coorientadora. III. Título.

CDU 519.6*32(043)

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

A utilidade dos valores referência de métricas na avaliação da qualidade de softwares orientados por objeto

PRISCILA PEREIRA DE SOUZA

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

PROFA. MARIZA ANDRADE DA SILVA BIGONHA - Orientadora
Departamento de Ciência da Computação - UFMG

PROFA. KÉCIA ALINE MARQUES FERREIRA - Coorientadora
Departamento de Computação - CEFET-MG

PROF. EDUARDO MAGNO LAGES FIGUEIREDO
Departamento de Ciência da Computação - UFMG

PROF. ROBERTO DA SILVA BIGONHA
Departamento de Ciência da Computação - UFMG

Belo Horizonte, 31 de outubro de 2016.

Agradecimentos

Quem vivenciou a trajetória do curso de mestrado sabe perfeitamente que é uma tarefa árdua. Mas, por outro lado, é uma experiência enriquecedora e de plena superação. A conclusão desta dissertação marca o fim desta importante etapa na minha vida. Gostaria, portanto, de agradecer àqueles que compartilharam comigo deste momento.

Agradeço primeiramente a Deus por ser sempre o meu guia, por iluminar os meus caminhos e principalmente a força nos momentos mais difíceis em que pensei em desistir. A conquista dessa vitória seria impossível sem Deus! De Deus vem tudo o que sei, tenho e sou. É Ele o meu maior mestre!

Agradeço à minha família por terem me ensinado o valor do conhecimento e das conquistas. Agradeço por me apoiarem em todas as minhas iniciativas, pela paciência, o grande incentivo e a compreensão nos momentos em que estive ausente.

Às minhas queridas orientadoras, professoras Mariza Bigonha e Kecia Ferreira, o empenho, conhecimento compartilhado e dedicação em realizar uma ótima orientação no mais alto nível de excelência. Muito obrigada por terem me recebido e acolhido de braços abertos.

Agradeço ao Programa de Pós-Graduação em Ciência da Computação (PPGCC) por me permitir realizar esse sonho!

Por fim, agradeço a todas as pessoas que contribuíram, de forma direta ou indireta, na conclusão deste trabalho e me deram o incentivo para sempre levá-lo adiante.

Que nunca se apaguem em mim a vontade de lutar, a esperança de vencer, a capacidade de resistir e a alegria de comemorar!

"Grandes coisas fez o SENHOR por nós,
e por isso estamos alegres."
(Sl. 126.3)

Resumo

Métricas de software são instrumentos para avaliação da qualidade de sistemas de software. Na literatura, há uma grande quantidade de métricas de software aplicáveis a sistemas implementados em diferentes paradigmas, como Programação Orientada por Objetos (POO). A fim de guiar a utilização dessas métricas na avaliação da qualidade de sistemas de software, é importante a definição de valores referência que mostram se o valor de uma métrica é aceitável. Utilizando valores referência, engenheiros de software podem identificar, por exemplo, pontos críticos para manutenção e evolução de sistemas. Diversos estudos propõem técnicas para derivar valores referência, porém, é necessária uma avaliação deles. Neste trabalho foi realizado um mapeamento sistemático da literatura com o objetivo de se identificar como valores referência de métricas de software têm sido aplicados, em particular, na detecção de *bad smells* e na predição de falhas. Esses dois aspectos foram considerados por estarem fortemente relacionados à qualidade interna do produto de software. Nesta dissertação, analisou-se a eficácia da utilização dos valores referência de métricas na avaliação da qualidade de sistemas de software orientados por objetos. Para realizar o estudo foi selecionado um catálogo de valores referência para 18 métricas de software derivados a partir de 100 sistemas de software. Foram definidas estratégias de detecção para cinco *bad smells* baseadas nos valores referência propostos nesse catálogo. A fim de aferir a eficácia dos valores referência derivados foram conduzidos dois estudos: (i) análise de detecção de cinco *bad smells* em 12 sistemas de software utilizando-se as estratégias propostas nesta dissertação e (ii) aplicação dos valores referência na predição de falhas em 10 sistemas de software. Os principais resultados deste trabalho mostram como valores referência de métricas de software podem ser empregados para a detecção de *bad smells* e para a predição de falhas.

Palavras-chave: Métricas de Software, Qualidade de Software, Valores Referência, Predição de Falhas, *Bad Smell*.

Abstract

Software metrics can be an effective measurement tool to assess the quality of software. In the literature, there are a lot of software metrics applicable to systems implemented in different paradigms like Objects Oriented Programming (OOP). To guide the use of these metrics in the evaluation of the quality of software systems, it is important to define thresholds. Using thresholds' values, software engineers may identify, for example, critical points for software maintenance and evolution. Several studies have proposed techniques for deriving threshold values; however, an evaluation of them is required. This master thesis presents a systematic mapping of the literature in order to identify how thresholds of software metrics have been applied, in particular, in detect bad smells and failure prediction. These two aspects were considered to be strongly related to the internal quality of the software product. This master thesis analyses the usefulness of the thresholds of object-oriented software metrics to evaluate the quality of software systems was selected. For this research, a threshold catalog of 18 software metrics derived from 100 software systems. In this master thesis, we have defined detection strategies for five bad smells based on the thresholds proposed in this catalog. The aim of this study is to investigate the effectiveness of the thresholds in: (i) detection analysis of 12 software systems using the strategies proposed in this master thesis and (ii) application of the thresholds to predict failure in 10 software systems. The main results of these studies show how thresholds of object-oriented software metrics can be used for the detection of bad smells and failure prediction.

Keywords: Software Metrics, Software Quality, Thresholds, Fault Prediction, Bad Smell.

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

CA Acoplamento Aferente

CE Acoplamento Eferente

DIT Profundidade de Árvore de Herança

LCOM Ausência de Coesão em Métodos

MLOC Linhas de Código por Método

NBD Profundidade de Blocos Aninhados

NOC Número de Classes

NOF Número de Atributos

NOM Número de Métodos

NORM Número de Métodos Sobrescritos

NSC Número de Filhas

NSF Número de Atributos Estáticos

NSM Número de Métodos Estáticos

PAR Número de Parâmetros

RMD Distância Normalizada

SIX Índice de Especialização

VG Complexidade de McCabe

WMC Métodos Ponderados por Classe

Sumário

Capítulo 1

Introdução

Uma questão central em Engenharia de Software é a produção de software de maneira eficiente, com qualidade, de fácil entendimento e manutenção, a um baixo custo e em um período de tempo aceitável. A utilização de práticas baseadas em objetos visa contribuir com isso, dentre outros aspectos, viabilizando implementar sistemas seguros, com mais qualidade, com um menor custo e menor tempo [Amber, 1998].

Problemas na qualidade de software geralmente são identificados por meio de testes e de inspeções. Apesar de essas atividades serem eficazes, elas demandam tempo e geram custos. Segundo Pressman [2006], métricas[1] de software proporcionam medidas quantitativas para avaliar a qualidade dos projetos de software, permitindo aos engenheiros de software efetuarem alterações no decorrer do processo de desenvolvimento a fim de promover a melhoria da qualidade do produto final. Os principais objetivos das métricas de software são: prover um melhor entendimento da qualidade do produto, avaliar a efetividade do processo e melhorar a qualidade do trabalho executado na fase de projeto. Mas, para realizar uma medição efetiva do software e elevar o uso de métricas para a tomada de decisão, é essencial a definição de valores referência significativos [Alves et al., 2010].

Apesar da importância das métricas no gerenciamento da qualidade de softwares orientados por objetos, elas ainda não são efetivamente utilizadas na indústria de software [Riaz et al., 2009; Tempero et al., 2010; Ferreira et al., 2012]. Uma possível razão é que, para a grande maioria das métricas, não são conhecidos os valores referência (do inglês, *thresholds*) [Ferreira et al., 2012]. Os valores referência auxiliam a controlar a qualidade interna e avaliar a facilidade de manutenção de software. Além disso, os

[1] A norma ISO/IEC 25010:2011 *(Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models)* utiliza o termo "medida". Todavia, para manter consistência com a literatura, que em sua maior parte emprega o termo “métrica”, utiliza-se também “métrica” nesta dissertação.

engenheiros de software também podem utilizar os valores referência para identificar, por exemplo, em quais pontos do sistema pode-se otimizar a alocação de recursos que podem ser fundamentais para o sucesso do projeto de software. Por exemplo, uma classe do sistema que apresenta um valor considerado alto para determinada métrica, dado o valor referência estabelecido para ela, pode ser refatorada, melhorando a qualidade interna de software e gerando impacto direto na sua qualidade externa. Da mesma forma, classes com valores de métricas considerados críticos podem receber maior atenção durante a fase de testes, otimizando o processo de manutenção e proporcionando a obtenção de um sistema de software com maior grau de corretude [Filó, 2014; Filó et al., 2015].

A definição de métodos para derivação de valores referência para métricas de software orientado por objetos tem sido o foco principal de diversos estudos realizados nesta área até então [Rosenberg et al., 1999; Benlarbi et al., 2000; Shatnawi et al., 2010; Alves et al., 2010; Chhikara et al., 2011; Ferreira et al., 2012; Kaur et al., 2013; Couto et al., 2013a; Oliveira et al., 2014; Filó, 2014; Filó et al., 2015; Vale et al., 2015]. Embora atualmente haja dezenas de métricas propostas na literatura e uma quantidade razoável de métodos definidos para derivação de valores referência, há poucas métricas com valores referência conhecidos, o que dificulta e até mesmo pode inviabilizar o gerenciamento da qualidade de software por meios quantitativos.

Ademais, embora os valores referência para métricas de software propostos na literatura tenham sido avaliados, tais avaliações não são abrangentes. Os trabalhos de Filó [2014] e Filó et al. [2015] são os que possuem o maior número de métricas com valores referência identificados, 18 no total. Porém, ele só analisou a relação de algumas destas métricas com dois *bad smells*: *God Class e Long Method*.

Dado esse cenário, é necessário um estudo empírico mais abrangente e detalhado para investigar a aplicabilidade de valores referências. É importante, dentre outros aspectos, investigar se os valores referência são úteis na identificação de *bad smells* e na predição de falhas, que são itens que impactam diretamente na qualidade de softwares orientados por objetos.

Tendo em vista tal necessidade, as seguintes questões de pesquisa direcionam este trabalho de dissertação:

QP1. *Os valores referência de métricas de software orientados por objetos auxiliam a identificar bad smells?*

QP2. *Os valores referência de métricas de software orientados por objetos auxiliam a predizer falhas em um software?*

1.1 Objetivo

O objetivo geral deste trabalho é: investigar a utilidade dos valores referência na avaliação da qualidade de softwares orientados por objeto; avançar no estado da arte analisando uma maior quantidade de métricas com valores referência; compreender e estabelecer possíveis relações entre valores referência de métricas com falhas e com *bad smells*.

1.1.1 Objetivos Específicos

Os objetivos específicos desta dissertação são:

- avaliação dos valores referência em softwares abertos orientados por objeto para verificar se por meio deles é possível realizar a detecção de *bad smells*;
- avaliação dos valores referência em softwares abertos orientados por objeto para verificar se por meio deles é possível realizar predição de falhas.

1.2 Contribuições

Neste estudo, foi avaliado o catálogo de valores referência apresentado por Filó et al. [2015], que contêm 18 métricas de software amplamente utilizadas. Esses valores referência foram derivados a partir de 100 sistemas de software reais. A técnica de derivação proposta por Filó et al. [2015] classifica as métricas em três faixas de valores, *Bom, Regular e Ruim*, de acordo com a frequência dos valores observados em um conjunto de sistemas.

A fim de aferir a qualidade dos valores referência derivados, foram conduzidos dois estudos. Tais estudos avaliam a eficácia dos valores referência (i) na detecção de cinco *bad smells* em 12 sistemas do *Qualitas.class Corpus* [Terra et al., 2013], utilizando-se estratégias de detecção propostas nesta dissertação e (ii) na predição de falhas em 10 sistemas de software do mesmo *corpus*.

Como resultado do trabalho apresentado nesta dissertação, destacam-se três principais contribuições:

- um mapeamento sistemático da literatura com foco em trabalhos sobre valores referência das métricas associados a *bad smells* e predição de falhas. Os resultados obtidos proporcionam à comunidade acadêmica uma visão do estado da arte nesse campo. Ao mesmo tempo revela a ausência de estudos com foco na utilização de valores referência para identificação de *bad smells* e predição de falhas.

- avaliação dos valores referência em softwares abertos com o objetivo de verificar se, de fato, é possível por meio deles realizar a detecção de *bad smells* em softwares orientados por objeto.

- avaliação dos valores referência em softwares abertos com o objetivo de verificar se, de fato, é possível por meio deles realizar a predição de falhas em sistemas de software orientados por objeto.

1.3 Estrutura da Dissertação

Esta dissertação está organizada conforme descrito a seguir.

Capítulo 2 detalha a fundamentação teórica necessária para a compreensão dos principais conceitos envolvidos nesse trabalho de pesquisa, a saber: qualidade de software, métricas de softwares orientados por objetos, valores referência para métricas de softwares orientados por objetos, métodos para a predição de falhas e identificação de *bad smells*.

Capítulo 3 descreve o planejamento e os resultados do mapeamento sistemático da literatura realizado para investigar a aplicação de valores referência de métricas de softwares orientados por objeto na detecção de *bad smell* e na predição de falhas.

Capítulo 4 apresenta a metodologia definida para execução deste trabalho.

Capítulo 5 descreve as estratégias de detecção de *bad smell* propostas nesta dissertação.

Capítulo 6 é dedicado à descrição dos experimentos realizados para verificar a possibilidade dos valores referências de métricas de softwares orientados por objeto detectarem *bad smells*. Esse estudo coloca os valores referência de Filó [2014] em perspectiva em relação a listas de referência geradas por ferramentas automatizadas de detecção de *bad smells*.

Capítulo 7 descreve outro estudo realizado para verificar a possibilidade dos valores referências de métricas de softwares orientados por objeto detectarem *bad smells*. Esse estudo coloca os valores referência de Filó [2014] em perspectiva em relação a listas de referência geradas por um especialista em programação orientada por objetos e *bad smells*

Capítulo 8 é dedicado à descrição dos experimentos realizados para verificar a capacidade dos valores referência de Filó [2014] predizerem falhas em sistemas de software.

Capítulo 9 conclui esta dissertação apresentando as observações finais e sugestões para trabalhos futuros.

Capítulo 2

Referencial Teórico

Este capítulo apresenta os principais conceitos aplicados no desenvolvimento desta dissertação. São eles: qualidade de software, métricas de softwares orientados por objetos, medição de software, valores referência para métricas de softwares orientados por objetos, predição de falhas e *bad smells*.

2.1 Qualidade de Software

Qualidade pode ser definida como "o grau em que características de um produto ou serviço de software satisfazem necessidades explícitas e implícitas dos *stakeholders*, agregando valor a esse produto ou serviço" [ISO/IEC 25010, 2011]. Qualidade interna é a totalidade das características do software do ponto de vista interno. Essa qualidade é mensurada e avaliada em relação à qualidade do código-fonte por meio de medidas de software [ISO/IEC 25010, 2011; Kayarvizhy & Kanmani, 2011]. Na ISO/IEC 25010, a qualidade interna é parte do modelo de qualidade de sistemas de software.

A preocupação com a qualidade dos produtos de software tem se tornado requisito imprescindível devido ao fato de o mercado estar cada vez mais competitivo, e ao fato de que organizações que disponibilizam produtos com baixa qualidade podem rapidamente perder seus clientes [Aranha & Borba, 2007]. Por essa razão, tem-se criado uma conscientização da importância do gerenciamento de qualidade de software e da utilização de técnicas de gerenciamento de qualidade provenientes da necessidade de realizar manutenção no software [Sommervile, 2011].

Um elemento fundamental para qualquer processo que objetive atingir qualidade em seu produto final é a medição. Medidas são utilizadas com o objetivo de que os modelos criados tenham sua qualidade avaliada e, consequentemente, para que o padrão de qualidade estabelecido como aceitável seja atingido no fim do processo. A

medição é vista como um trabalho pragmático com o objetivo de encontrar o indicador direto para um determinado aspecto da qualidade, no entanto, sem critérios claros, as medições podem impedir comparações úteis com outros projetos [Baggen et al., 2012]. Portanto, métricas têm de ser escolhidas com base em uma referência clara e confiável. Elas são experimentalmente avaliadas para medir um atributo de qualidade [Baggen et al., 2012].

Na próximas seções é possível observar que cada vez mais, as métricas de software têm sido propostas como mecanismos para avaliar a qualidade [Chidamber & Kemerer, 1994; Marinescu, 2004].

2.2 Métricas de Software

Medição é parte fundamental de qualquer disciplina de engenharia, e disciplinas da Engenharia de Software não são exceções. Sommervile [2011] define métrica como um valor numérico relacionado a algum atributo de software, permitindo a comparação entre atributos da mesma natureza. Ele argumenta que a medição permite realizar previsões gerais de um sistema e identificar componentes com anomalias.

As métricas de software referem-se a medições que podem ser aplicadas para verificar os indicadores de processos, projetos e produtos de software. As métricas possibilitam medições de padrões para avaliar determinados atributos que estejam relacionados ao software. De acordo com Sommervile [2011] por meio das métricas é possível gerenciar um software. É possível também, por exemplo, via atributos internos de um software, definir atributos externos como facilidade de manutenção, confiabilidade, portabilidade e facilidade de uso.

As métricas também podem ser usadas para identificar problemas estruturais nas fases iniciais do ciclo de vida do software e na identificação dos desvios de projeto, conhecidos na literatura como *bad smells* [Fowler, 1999]. Alguns trabalhos têm sido desenvolvidos para identificar *bad smells* em software com o uso de métricas a partir de código-fonte [Marinescu, 2002; Lanza & Marinescu, 2007], e no início do projeto de desenvolvimento por meio de diagramas de classes [Nunes, 2014].

Existem diversas métricas para orientação por objetos propostas na literatura, dentre as quais destacam-se:

- Métricas CK: propostas por Chidamber & Kemerer [1994], elas avaliam o software orientado por objetos no nível de classe. Os aspectos avaliados por essas métricas são: acoplamento, herança e coesão. Existem seis métricas CK: Métodos Ponderados por Classe (WMC), Profundidade de Árvore de Herança (DIT), Nú-

mero de Filhos (NOC), Acoplamento entre Classes de Objetos (CBO), Resposta de Classe (RFC), Ausência de Coesão em Métodos (LCOM).

Chidamber & Kemerer [1994] afirmam que esse conjunto de métricas pode auxiliar os usuários no entendimento da complexidade de um projeto orientado por objetos em relação aos atributos externos de qualidade como manutenção e reutilização de software e na detecção de anomalias. Tais métricas foram propostas com o intuito de avaliar atributos de qualidade externa do software, tais como: defeitos de software, testes e esforço de manutenção [Basili et al., 1996; Chidamber & Kemerer, 1994].

A seguir, destaca-se do conjunto de métricas CK orientadas por objetos apenas aquelas que possuem valores referência disponíveis na literatura.

- WMC (*Weighted Methods per Class*): é uma métrica que representa a complexidade de uma classe por meio de seus métodos. O cálculo é realizado via atribuição de pesos aos métodos da classe. Pode-se ponderar os métodos por linhas de código, complexidade ciclomática, número de parâmetros, etc. Quanto maior o valor para essa métrica, mais complexa a classe. Segundo Chidamber & Kemerer [1994], essa métrica é um indicador do custo de desenvolvimento e manutenção de uma classe, assim como do grau de reúso da classe.
- DIT (*Depth of Inheritance*): indica a posição de uma classe na árvore de herança de um software, que é dada pela distância máxima da classe até a raiz da árvore. Essa métrica é considerada um indicador da complexidade de desenho e de predição do comportamento de uma classe pois, quanto maior a profundidade da herança, mais complexo é o projeto de software e mais difícil é para se entender um módulo. Muitas classes de objeto podem precisar ser compreendidas para que as classes de objetos nas folhas das árvores sejam entendidas.
- NOC (*Number of Children*): é uma medida do número de subclasses imediatas de uma classe. É um indicador da importância que uma classe tem no sistema, pois quanto mais subclasses ela possuir, maior a importância de seu teste no sistema. Um valor alto para NOC pode indicar mais esforço para validação de classes de base por causa do número de subclasses que dependem delas.
- LCOM (*Lack of Cohesion of Methods*): é uma métrica que indica o quanto os métodos de uma classe acessam atributos em comum. De acordo com

Chidamber & Kemerer [1994], a coesão entre os métodos de uma classe é definida pela similaridade entre eles. A avaliação da similaridade entre dois métodos é determinada pela diferença entre o número de pares de métodos sem atributos compartilhados e o número de pares de métodos com atributos compartilhados.

- Métricas MOOD: propostas por Abreu & Carapuça [1994], são compostas por oito métricas de software orientado por objetos que avaliam aspectos de herança, encapsulamento, coesão, acoplamento, polimorfismo e reúso de software. O conjunto MOOD é composto das seguintes métricas: Fator Ocultação de Método, Fator Ocultação de Atributo, Fator Herança de Método, Fator Herança de Atributo, Fator Acoplamento, Fator Polimorfismo e Fator Reúso.

  O conjunto MOOD tem por objetivo proporcionar indicadores quantitativos para as propriedades dos projetos orientados por objeto via métricas [Pressman, 2006]. O escopo dessas métricas é o sistema.

  A única métrica do conjunto MOOD que possui valor referência disponível na literatura é a métrica COF descrita a seguir:

  - COF (*Coupling Factor*): é dada pela razão entre o total de conexões entre classes existentes no software e o total de conexões possíveis entre elas. As conexões representam os relacionamentos entre as classes, que podem ocorrer. Esse relacionamento pode ser pela passagem de parâmetros ou por referência direta de um atributo ou método de uma classe por outra. O resultado obtido permite avaliar o quão acopladas estão as classes do sistema.

- Métricas MARTIN: Martin [1994] propõe um conjunto de métricas para medir a qualidade de softwares orientados por objetos em termos de acoplamento. O escopo dessas métricas é o pacote. Projetos com alto grau de acoplamento tendem a ser rígidos, não reutilizáveis e de baixa qualidade em relação à manutenabilidade. O conjunto é composto pelas seguintes métricas: Acoplamento Aferente (AC), Acoplamento Eferente (EC), Instabilidade, Abstração e Distância Normalizada.

  Do conjunto de métricas Martin, três possuem valores referência disponíveis na literatura. São elas:

  - AC (*Afferent Coupling*): classes podem ser agrupadas em categorias, formando grupos coesos. O acoplamento aferente conta o número de classes fora

da categoria que dependem de classes dentro da mesma categoria. Quanto maior o AC, maior a quantidade de classes às quais a categoria provê serviços, indicando sua relevância dentro do projeto. Valores altos de AC indicam que o projeto possui alto grau de acoplamento.

- EC (*Efferent Coupling*): o acoplamento eferente conta o número de classes fora da categoria que dependem de classes dentro da mesma categoria. Essa métrica é um indicativo do nível de acoplamento da categoria às outras classes do projeto. Um valor alto de EC indica que a categoria consome mais serviços providos por classes externas à categoria, o que corresponde a um alto grau de acoplamento.
- *Normalized Distance* (RMD = $|A + I + 1|$): essa métrica mede a distância perpendicular de instabilidade (I) e abstração (A) da *Main Sequence* (representa o balanceamento entre abstração e instabilidade). Ou seja, a métrica indica o quão longe a relação está do ponto mais próximo do grau de balanceamento considerado entre instabilidade e abstração. Quanto maior for essa distância, menor o grau de balanceamento entre instabilidade e abstração.

- Métricas de Complexidade: são métricas utilizadas na literatura como indicadores de complexidade nos softwares. Destacam-se aqui, do conjunto de métricas de complexidade orientadas por objetos, somente aquelas que possuem valores referência disponíveis na literatura:

  - MLOC (*Method Lines of Code*): essa métrica conta o número de linhas de código de um método.
  - SIX (*Specialization Index*): proposta por Lorenz e Kidd [1994] como a razão do número de métodos sobrescritos na classe avaliada, ponderado pela métrica DIT da classe avaliada, sobre o número de métodos da classe. O objetivo dessa métrica é avaliar o quanto uma classe sobrescreve o comportamento de suas superclasses. Quanto maior o valor do índice de especialização, maior é a execução de comportamentos especializados em tempo de execução, o que indica que a classe é complexa e deve receber um maior esforço de testes.
  - NBD (*Nested Block Depth*): esta métrica mede a profundidade máxima de blocos aninhados no sistema. Ela é um indicativo da complexidade do sistema, uma vez que conforme a quantidade de blocos aninhados aumenta, o trecho de código torna-se mais difícil de entender e consequentemente mais difícil é a sua manutenção.

  - VG (*McCabe Cyclomatic Complexity*): métrica aplicada em métodos, proposta por McCabe [1976]. Via medição das alternativas possíveis no controle de fluxo de um método ou classes, é feita uma estimativa da legibilidade do código. A cada divisão de fluxo do código, como em, *if, for, while, do, case, catch*, operador ternários e operadores condicionais lógicos, a métrica é acrescida de 1.

- Métricas de Lorenz e Kidd: As métricas de Lorenz & Kidd [1994] são definidas para classes e são divididas em quatro categorias principais: tamanho (baseada na contagem de atributos e operações da classe individualmente e na média para um valor do sistema como um todo), herança (avaliação de como as operações são reutilizadas via hierarquia de classe), atributos internos (avaliação de coesão) e externos (acoplamento e reutilização) [Pressman, 2006]. As métricas são as seguintes:

  - NCA (*Number of Afferent Connections*): a métrica NCA verifica a influência da classe avaliada sobre as outras classes do software, ou seja, a quantidade de classes que usam os serviços da classe avaliada.
  - NMP (*Number of Public Methods*): a métrica NMP permite avaliar o tamanho de uma classe e a quantidade de serviços dela.
  - NAP(*Number of Public Attributes*): a métrica NAP define quantos atributos de uma classe são públicos.
  - NOF (*Number of Attributes*): número total de atributos em um escopo selecionado.
  - NOM (*Number of Methods*): mede o número de métodos de cada classe.
  - NORM (*Number of Overridden Methods*): indica a quantidade de métodos redefinidos de uma classe.
  - NSC (*Number of Children*): número de subclasses diretas de uma classe.
  - NSF (*Number of Static Attributes*): mede o número de atributos estáticos de uma classe.
  - NSM (*Number of Static Methods*): número total de métodos estáticos de uma classe.
  - PAR (*Number of Parameters*): número total de parâmetros de um método.

  De acordo com Sommervile [2011], avaliar a qualidade do software por meio de medições possibilita definir quantitativamente o sucesso ou a falha de determinado atributo, identificando a necessidade de melhorias. Gerenciar a qualidade

do software pode permitir que sejam alcançados um baixo número de defeitos e padrões controláveis de manutenibilidade, confiabilidade e portabilidade.

Apesar da importância das métricas no gerenciamento da qualidade de softwares orientados por objetos, elas ainda não foram efetivamente utilizadas na indústria de software [Riaz et al., 2009; Tempero et al., 2010; Ferreira et al., 2012; Filó, 2014; Filó et al., 2015]. Uma possível razão é que para a grande maioria das métricas, não são conhecidos valores referência Ferreira et al. [2012]. Alguns trabalhos têm sido realizados recentemente para contornar esse problema. Esse tópico é tratado na Seção 2.3. Além disso, tem sido investigado na literatura a relação dos valores referência na predição de falhas e na detecção de *bad smells*. Uma visão geral dessas aplicações será fornecida nas Seções 2.4 e 2.5.

2.3 Valores Referência

Valores referência são valores obtidos por observação ou mensuração que auxiliam na avaliação da qualidade do software [Benlarbi et al., 2000]. Com a utilização de valores referência para métricas de software orientado por objetos, acredita-se que possíveis problemas no software podem ser identificados durante o projeto, resultando em um software de maior qualidade e, consequentemente, com um menor custo de manutenção. Além disso, informações acerca da qualidade interna do software podem ser obtidas e direcionar esforços de testes e manutenções preventivas para pontos críticos do sistema, otimizando a aplicação dos recursos disponíveis durante a fase de operação, fase em que o sistema é utilizado efetivamente pelos usuários [Riaz et al., 2009; Tempero et al., 2010; Ferreira et al., 2012].

A literatura apresenta alguns trabalhos que vem sendo realizados com o objetivo de derivar valores referência para métricas [Rosenberg et al., 1999; Benlarbi et al., 2000; Shatnawi et al., 2010; Chhikara et al., 2011; Ferreira et al., 2012; Kaur et al., 2013; Oliveira et al., 2014; Filó, 2014; Filó et al., 2015; Vale et al., 2015]. Esses trabalhos variam principalmente na metodologia de pesquisa utilizada para estabelecer esses valores, que de acordo com Filó [2014] podem ser categorizadas da seguinte forma:

- Pesquisa Quantitativa: usam técnicas estatísticas por meio das seguintes estratégias:

    * Regressão Logística: utiliza modelos de regressão logística com o intuito de identificar grupos de risco, que são classes fora dos valores definidos, a partir de um conjunto de dados constituídos por métricas.
    * Análise de Distribuição: baseiam sua análise em distribuições estatísticas da coleta de medições de um grande número de softwares.
    * Análise ROC (*Receiver-Operating Characteristic*): utiliza uma representação gráfica para ilustrar e avaliar a correta avaliação da classe em relação à presença de erros com a variação do valor referência.

  – Pesquisa Qualitativa: avaliam projetos propostos para um mesmo problema, partindo de modelagens sabidamente ruins para identificar a evolução das métricas.

Com a aplicação das métricas em conjunto com um catálogo de valores referência, as avaliações podem ser realizadas de forma automatizada, baseada em aspectos quantitativos do software, auxiliando na garantia da qualidade.

Avaliar a eficácia da utilização dos valores referência propostos na literatura é um ponto importante de ser investigado. Em sua maior parte, os valores referência propostos não foram avaliados amplamente, mas avaliá-los é de fundamental importância para a sua aplicação na prática. Neste trabalho de dissertação, a utilidade dos valores referência é estudada do ponto de vista de identificação de falhas e de *bad smells*.

2.3.1 Identificação de Valores Referência

Esta seção descreve alguns trabalhos mais recentes para a identificação de valores referência de métricas de software.

Alves et al. [2010]

Alves et al. [2010] projetaram um método que determina valores referência empiricamente a partir de um conjunto de softwares e suas medições. No método proposto, (i) atribui-se um peso às entidades do software, baseado nas linhas de código, (ii) agrega-se o peso das entidades de acordo com os valores das métrica (ii) os pesos são normalizados de acordo com os sistemas, garantindo que a soma das porcentagens dos pesos das entidades continue 100% e, (iv) agrega-se o peso das entidades de acordo com os valores das métricas, dessa vez, para todos os sistemas. Os valores das métricas são, então, ordenados em forma ascendente, estabelecendo-se os valores referência

por meio da escolha dos 70%, 80% e 90% percentis desses dados, derivando os seguintes perfis de qualidade: baixo risco (entre 0 - 70%), risco moderado (70 - 80%), alto risco (80 - 90%) e altíssimo risco ($>$ 90%). Os autores focam o trabalho na descrição do método, aplicando-o na derivação de valores referência para 3 métricas em nível de métodos e duas métricas em nível de classe.

Herbold et al. [2011]

Herbold et al. [2011] definem um método algorítmico para calcular e otimizar os valores referência de um conjunto de métricas. Nesse trabalho foram utilizadas as técnicas de aprendizado de máquina e de mineração de dados. De acordo com os autores, a metodologia proposta é independente do conjunto de métricas e, portanto, também independente de qualquer linguagem de programação, paradigma ou nível de abstração. Quatro estudos de caso foram realizados em softwares de código aberto para validar a metodologia, eles possuem funções em C, C ++, métodos C# e classes Java. Em dois dos estudos de casos, o método foi capaz de melhorar significativamente a eficiência dos conjuntos de métricas existentes. Nos dois outros estudos de caso, foi possível reproduzir com sucesso classificações complexas utilizando valores referência simples.

Ferreira [2011]

O trabalho de Ferreira [2011] determina valores referência para seis métricas de software: LCOM (*lack of cohesion in methods*), DIT (*depth in inheritance tree*) [Chidamber & Kemerer, 1994], COF (*coupling factor*) [Abreu & Carapuça, 1994], Conexões Aferentes, Número de Métodos Públicos e Número de Atributos Públicos. Os resultados da análise realizada nesse trabalho mostram que os valores dessas métricas não seguem uma distribuição normal. Foi então identificada uma distribuição de probabilidades adequada aos valores de cada uma dessas métricas. Além disso, foi apresentado o método utilizado para derivação de valores referência de métricas baseado em análise estatística dos dados.

Ferreira et al. [2012]

Ferreira et al. [2012] reportam os valores referência em intervalos de valores constituídos por três faixas, denominadas: *Bom*, representando os valores mais frequentes para métricas em sistemas de software de boa qualidade; *Regular*, correspondendo a valores pouco frequentes para métricas em sistemas de software de boa qualidade; *Ruim*, correspondendo a valores raros para métricas

em sistemas de software de boa qualidade. A ideia das faixas sugeridas como valores referência pelos autores é que, uma vez que os valores são frequentes em sistemas de software, isso indica que eles correspondem à prática comum no desenvolvimento de software de alta qualidade, o que serve como um parâmetro de comparação de um software com os demais. Da mesma forma, os valores pouco frequentes indicam situações não usuais na prática, portanto, pontos a serem considerados como críticos.

Zhang et al. [2013]

Zhang et al. [2013] mostram que os valores referência dependem do contexto e, portanto, não podem ser generalizados a todos os tipos de sistemas de software. Em particular, a linguagem de programação ou o domínio de aplicação são contextos que têm um forte impacto sobre os valores referência. Isso fortalece ainda mais os resultados obtidos por Nagappan et al. [2006] que mostraram que os valores referência obtidos via a realização de uma análise de correlação são verdadeiros apenas para um conjunto limitado de sistemas de software similares.

Foucault et al. [2014]

O trabalho de Foucault et al. [2014] propõe uma abordagem com o objetivo de chegar a um *trade-off* entre a representatividade dos valores referências e eficiência do seu custo de computação. A abordagem é baseada em uma seleção imparcial de entidades de software e não faz suposições sobre as propriedades estatísticas de métricas de software. De acordo com os autores, a abordagem proposta pode ser usada por qualquer pessoa que queira rapidamente calcular um valor referência representativo para uma determinada métrica e um determinado contexto. A proposta é baseada na abordagem de Chidamber & Kemerer [1994]. Os autores argumentam que um percentual é adequado para computar valores referência para um dado contexto pois, ele é independente da distribuição das métricas e pode ser estimado em uma pequena amostra de entidades de software.

Filó [2014]

Filó [2014] apresenta um método de extração de valores referência para métricas de softwares orientados por objetos, bem como a sua aplicação na identificação de valores referência para 18 métricas em um *dataset* de 111 softwares de código aberto. Os autores propõem valores referência para um conjunto de métricas de softwares orientados por objetos por meio de três faixas de valores:

Bom/Frequente, Regular/Ocasional e Ruim/Raro. A faixa *Bom/Frequente* corresponde aos valores com alta frequência, caracterizando aos valores mais comuns da métrica, na prática. A faixa *Ruim/Raro* corresponde aos valores com baixa frequência, e a faixa *Regular/Ocasional* é intermediária, correspondendo aos valores que não são nem muito frequentes nem raros. Por exemplo, para a métrica Número de Métodos (NOM) têm-se os seguintes valores: *Bom/Frequente* (NOM = 6), *Regular/Ocasional* ($6 < \text{NOM} = 14$) e *Ruim/Raro* (NOM > 14). Os valores referência não expressam necessariamente as melhores práticas da Engenharia de Software, e sim um padrão seguido pela maioria dos softwares. O método proposto não usa técnicas estatísticas muito complexas e é reprodutível.

Oliveira et al. [2014]

Oliveira et al. [2014] propõem o conceito de valores referência relativos para avaliar métricas cujos dados seguem distribuições cauda-pesada. Os valores referência propostos são chamados relativos pois assumem que os limites devem ser seguidos pela maioria das entidades de código-fonte. Foi proposto um método que extrai valores referência relativos de dados de um *benchmark*. Esse método foi avaliado na *Qualitas.class Corpus 2013* em 106 sistemas. De acordo com os autores, os valores referência extraídos representam um interessante equilíbrio entre as regras de projeto reais e as idealizadas. Além disso, Oliveira et al. [2014] propõem uma ferramenta de código aberto capaz de extrair valores referência a partir dos dados de medição de um *benchmark* de sistemas de software. De acordo com os resultados apresentados, a ferramenta tem sido aplicada com êxito em uma coleção de softwares abertos desenvolvidos em Java.

Vale et al. [2015]

Vale et al. [2015] compara três métodos para derivar valores referência em um *benchmark* de 33 SPLs *(Software Product Lines)*: O método de Alves et al. [2010], o método de Ferreira et al. [2012] e o método de Oliveira et al. [2014]. Vale et al. [2015] avaliam quais desses métodos derivam valores referência apropriados para quatro métricas utilizadas em linha de produto de software: *Lines of Code (LoC), Coupling between Objects classes (CBO), Weighted Method per Class (WMC), and Number of Constant Refinements (NCR)*. Os valores referência obtidos foram usados para a identificação do *bad smell God Class*.

Vale [2016]
O estudo de Vale [2016] propõe um método para calcular valores referência para métricas chamado *Vales method*. Esse método contempla todos os pontos identificados no estudo de Vale et al. [2015] que de acordo com os autores são considerados desejáveis para um método de derivação de valores referência. Para avaliar o método proposto, os valores referência foram analisados individualmente e foi utilizada uma estratégia baseada em métricas. Os resultados foram analisados por meio de duas bases de dados com métricas de diversos sistemas. O método proposto foi comparado com os métodos usados para comparação no estudo de Vale et al. [2015].

Nesta dissertação será utilizado o catálogo de valores referência definido por Filó [2014], pois (i) os valores referência propostos por Filó [2014] consideram a frequência de vários softwares e não simplesmente a média dos valores, (ii) os valores referência foram avaliados empiricamente (iii) este catalogo é o que possui o maior número de valores referência propostos dentre os trabalhos encontrados na literatura e (iv) Filó [2014] efetuou uma avaliação de seus valores referência com *bad smells*, mas avaliou apenas dois *bad smells*, *God Class e Long Method*, não cobrindo todo o catálogo proposto.

2.4 Predição de Falhas de Software

A terminologia padrão para Engenharia de Software do *IEEE Institute of Electrical and Electronics Engineers* [Radatz et al., 1990] define os termos *erro, defeito e falha* da seguinte maneira:

- erro: engano, alguma coisa feita por humanos. Por exemplo, um desenvolvedor implementou um requisito de forma diferente do que fora especificado no documento.
- defeito: o resultado de um erro. Por exemplo, a presença de um requisito implementado de forma diferente da sua especificação.
- falha: diferença indesejável entre o observado e o esperado, software diferente do que é esperado pelo usuário. O usuário ao executar a aplicação, percebeu que os resultados esperados não coincidem com os resultados retornados.

Neste trabalho, utilizamos o termo *falha* visto que ele é frequentemente usado na literatura em trabalhos correlacionados com métricas, valores referência e

bad smells. Sommervile [2011] define falha de software como a incapacidade do software de realizar uma função requisitada, aspecto externo. Predição de falhas é uma área de pesquisa em Engenharia de Software que objetiva identificar os componentes de um sistema de software que são mais prováveis de apresentar defeitos [Couto et al., 2013a].

As técnicas de predição de falhas envolvem a análise do código-fonte de uma versão de um sistema [Basili et al., 1996] ou de dados históricos do controle de versão [Nagappan et al., 2006] e de sistemas de acompanhamento de defeitos [Kim et al., 2007]. A finalidade é prever a ocorrência ou o número de defeitos que serão encontrados em cada componente do sistema, por exemplo: pacote, classe, método ou qualquer outra unidade de código [D'Ambros et al., 2010].

Dentre os trabalhos encontrados na literatura que correlacionam predição de falhas e métricas de software, está o trabalho de Gyimothy et al. [2005] que investiga como a predição de falhas em um código-fonte do navegador *web Mozilla* pode ser realizada utilizando-se as métricas CK. Catal & Diri [2009] propõem um agrupamento de valores referência de métricas baseado em abordagens de previsão falhas de software quando os rótulos de falhas para os módulos estão indisponíveis. Couto et al. [2013a] propõem uma abordagem para predição de falhas centrada em evidências de causalidade entre métricas de código-fonte (como preditor) e a ocorrência de defeitos. Abaei et al. [2015] apresentam uma abordagem de predição de falhas para ser utilizada na ausência de dados de defeitos.

2.5 *Bad Smells*

Bad smells são um conjunto de más práticas de projeto popularizadas no livro *Refactoring* de Martin Fowler [1999]. De acordo com Fowler [1999], o termo *bad smell* (mau cheiro) se refere às características encontradas nas estruturas de códigos que indicam que eles podem estar com problemas e precisam ser reestruturados de alguma maneira. Os *bad smells* não são a causa direta de falhas na aplicação, mas podem influenciar indiretamente para a inserção de erros responsáveis por futuras falhas. Em geral, eles são responsáveis pelas dificuldades de manutenção, evolução e realização de testes no sistema e propicia a inserção de defeitos [Mansoor et al., 2014]. Por essa razão, é essencial saber como identificá-los para se aplicar os mecanismos necessários para sua remoção e, consequentemente, melhorar a qualidade do software.

Além de definir o termo *bad smell*, Fowler [1999] apresenta uma lista com 22 *bad smells* que aborda refatoração em código-fonte, mas não usa estratégias de detecção para reconhecê-los. O autor descreve características de como identificar essas anomalias em código-fonte, sem indicar quais metodologias poderiam ser utilizadas para tal tarefa.

A seguir são descritos os 22 *bad smells* identificados por Fowler [1999]:

- *Divergent Change*: essa anomalia ocorre quando uma classe é frequentemente modificada de diferentes formas, por diferentes razões.
- *Large Class*: corresponde a classes com alta responsabilidade no sistema. Essa anomalia pode ser descrita como um objeto que sabe demais ou faz muito. Ela representa uma classe que cresce muito para se tornar a classe que faz quase tudo no sistema.
- *Long Method*: são métodos que implementam várias funcionalidades, o que os tornam difíceis de serem mantidos e entendidos. *Long Method* tende a centralizar as funcionalidades de uma classe da mesma forma que a *Large Class* centraliza as funcionalidades de um subsistema inteiro, ou até mesmo de um sistema.
- *Feature Envy*: refere-se a uma classe que usa os métodos de outra classe em excesso. É um sinal de que um trecho de código está mal localizado e que deveria ser transferido para outra classe, ou seja, ele implementa um interesse semelhante ao de outra classe.
- *Shotgun Surgery*: são classes que ao serem alteradas causam impacto em outras classes. Uma classe com *Shotgun Surgery* é identificada quando ao realizar uma mudança nela, é gerada uma série de pequenas mudanças em muitas classes diferentes.
- *Long Parameter List*: são métodos com uma longa lista de parâmetros. O ideal é que as listas de parâmetros sejam curtas, pois quanto mais extensas se tornam, mais difíceis de ler e de usar.
- *Data Class*: esse *bad smell* corresponde a classes que não definem funcionalidades próprias.
- *Data Clumps*: refere-se a um conjunto considerável de atributos ou parâmetros que são usados com frequência em conjunto.
- *Parallel Inheritance Hierarchies*: ocorre nos casos em que toda vez que for feita uma subclasse de uma superclasse, terá que ser feita também uma subclasse de outra classe.

- *Duplicated Code*: considerado o mais comum dos *bad smells*. É caracterizado quando um código idêntico ou muito semelhante existe em mais de um local no código-fonte.
- *Primitive Obsession*: refere-se ao uso de muitas variáveis primitivas para representar tipos complexos. Se o tipo de dados é bastante complexo, o ideal é escrever uma classe para representá-lo.
- *Switch Statements*: um dos problemas do uso de sentenças com *switch* em orientação por objetos é a duplicação. Ao se adicionar uma nova cláusula ao *switch*, poderá ocorrer a necessidade de pesquisa em todas as sentenças com *switch* para a modificação das mesmas.
- *Lazy Class*: cada classe criada no sistema de software tem um custo para ser mantida e evoluída. *Lazy classes* são definidas como classes que não estão fazendo o suficiente para pagarem seu próprio custo.
- *Speculative Generality*: esse problema ocorre quando se projeta uma solução genérica para uma situação que possivelmente nunca ocorrerá, tornando objetos flexíveis o suficiente para tratar qualquer tipo de situação, por exemplo, criando-se classes abstratas.
- *Temporary Field*: é evidenciado quando a variável está no escopo da classe, quando deveria estar no escopo do método. Esse *bad smell* viola o princípio ocultação de informação.
- *Message Chains*: refere-se a presença de diversas chamadas semelhantes no código-fonte.
- *Middle Man*: ocorre quando uma classe recebe chamadas e utiliza um objeto interno para realizar a maioria de suas funções (delegação).
- *Inappropriate Intimacy*: refere-se a classes que usam os atributos internos e métodos de outra classe.
- *Alternative Classes with Different Interfaces*: ocorre com métodos que fazem a mesma coisa, mas com diferentes assinaturas.
- *Incomplete Library Class*: ocorre quando os programadores não consultam adequadamente as bibliotecas de classes, e acabam criando seus próprios algoritmos o que gera dois problemas: os programadores perdem tempo criando algo que se encontra pronto, e eles deixam de cooperar entre si.
- *Refused Bequest*: uma classe herda atributos e métodos de outra classe, mas não os usa. Isso indica que algo está errado com a decomposição hierárquica, classe e subclasses.

– *Comments*: comentários não são uma coisa ruim. Entretanto, excesso de comentários pode indicar que os nomes de métodos/atributos não estão suficientemente expressivos.

É importante que a remoção dos *bad smells* seja realizada o mais cedo possível no desenvolvimento do software. Entretanto, a identificação deles pode ser fragilizada, pois dependem diretamente da interpretação e habilidade de identificação do desenvolvedor. Outro problema é não conhecer os mecanismos que podem ser aplicados para a remoção desses *bad smells* de código. As métricas podem auxiliar a identificar *bad smells*. O trabalho de Marinescu [2002] é possivelmente um dos primeiros a definir estratégias de detecção de *bad smells* utilizando métricas. Lanza et al. [2006] também definem uma lista de *bad smells*, alguns novos e outros modificados a partir da obra de Fowler [1999], e usam estratégias de detecção para identificá-los. Em seu trabalho sobre refatoração, Hamza et al. [2008] descrevem os *bad smells* das obras de Fowler [1999] e Kerievsky [2005]. Outros trabalhos propõem a detecção de *bad smell* a partir de diagramas UML e diagramas de classes [Nunes, 2014; Bertrán, 2009] e a detecção de *bad smells* por meio de métricas de interesse Padilha et al. [2013].

Outro trabalho realizado nessa área é o de D'Ambros et al. [2010]. Os autores consideram que a maior parte dos problemas de software estão relacionados à manutenção e que métricas somente serão efetivas se forem combinadas formando estratégias de detecção. D'Ambros et al. [2010] também propõem relacionar *bad smells* com falhas de software.

2.6 Considerações Finais

As métricas têm papel fundamental na avaliação da qualidade de software, pois fornecem aos engenheiros de software dados que lhes permitem avaliar os processos de desenvolvimento, projetos e produtos, bem como controlá-los, melhorá-los e planejá-los. Além disso, elas podem auxiliar na identificação de desvios de projeto, conhecidos na literatura como *bad smells*. As métricas são fundamentais para avaliação quantitativa da presença de determinado atributo relacionado ao software.

Há uma grande quantidade de métricas de software orientado por objetos propostas na literatura. As mais referenciadas são as do conjunto CK [Chidamber & Kemerer, 1994], que define métricas para avaliar classes e o conjunto de métricas MOOD [Abreu & Carapuça, 1994] que define métricas para avaliar sistemas. Neste capítulo foi descrito o conceito de medição e de métricas e foi realizado um levantamento das principais

métricas existentes na literatura. Além disso, evidenciou-se a importância dos valores referência de métricas de softwares para viabilizar o uso de métricas na prática.

A predição de falhas é um recurso que pode ser utilizado para a obtenção de softwares mais confiáveis. A abordagem empregada em alguns dos trabalhos relacionados com esse tópico consiste em investigar correlação entre métricas de software e falhas no software.

Bad smells indicam possíveis anomalias em projetos de softwares. Para que as métricas possam identificar *bad smells* é necessário encontrar algo que os relacione. Esse relacionamento é possível por meio de valores referência, que definem valores numéricos que determinam quantitativamente se o valor de uma métrica é aceitável. O objetivo deste trabalho é investigar se os valores referência de métricas propostos na literatura auxiliam a identificação de *bad smells* e a predição de falhas.

O próximo capítulo apresenta um mapeamento sistemático da literatura sobre aplicação de valores referência na identificação de *bad smells* e na predição de falhas.

Capítulo 3

Mapeamento Sistemático da Literatura

Este capítulo apresenta um mapeamento sistemático da literatura com o objetivo de encontrar e analisar trabalhos primários relevantes e reconhecidos sobre aplicação de valores referência de métricas de software na garantia da qualidade de software, mais especificamente, na identificação de *bad smells* e na predição de falhas.

A Revisão Sistemática da Literatura (RSL) e Estudos de Mapeamentos Sistemáticos (EMS) surgiram como duas importantes ferramentas para agregar e construir conhecimento em engenharia de software [Kitchenham & Charters, 2007]. Kitchenham & Charters [2007] definem revisão sistemática da literatura "como um meio de identificar, avaliar e interpretar todas as evidências disponíveis relevantes para uma questão específica, área temática, ou fenômeno de interesse". Eles definem mapeamento sistemático "como uma ampla revisão dos estudos primários em um tema específico que visa identificar as evidências disponíveis sobre o tema" [Kitchenham & Charters, 2007].

De acordo com Kitchenham & Charters [2007] um estudo primário é um estudo experimental que investiga uma questão de pesquisa específica. O estudo secundário é um estudo que revisa todos os estudos primários relacionados a uma questão de pesquisa específica, contribuindo para integrar/sintetizar evidências sobre a questão de pesquisa. O mapeamento sistemático é uma forma de estudo secundário que realiza uma ampla revisão de estudos primários em um tópico, identificando evidências sobre o tópico. Por meio dele é possível identificar lacunas onde novos estudos primários são necessários e realizar agrupamentos de evidências que podem levar a novas revisões sistemáticas.

Kitchenham & Charters [2007] citam diversas diferenças entre Revisão Sistemática da Literatura e Mapeamento Sistemático, tais como: estudo de mapeamento geral-

mente têm questões de pesquisa mais genéricas e, muitas vezes, tratam questões de múltiplas pesquisas; os termos utilizados no processo de busca nos estudos de mapeamento são menos focados do que o processo de extração de dados de revisões sistemáticas; a etapa de análise de um mapeamento não incluem técnicas de análise aprofundada, como a meta-análise e síntese narrativa, mas apresentam totais e resumos; a divulgação dos resultados de um estudo de mapeamento pode ser mais limitada do que para uma revisão sistemática. Além disso, um mapeamento sistemático pode ser visto como uma alternativa complementar a uma revisão sistemática, no sentido de poder ser usado quando as evidências empíricas em uma determinada área são poucas, ou se o tópico de pesquisa for muito amplo para uma revisão sistemática ser viável [Kitchenham & Charters, 2007].

O mapeamento sistemático mostrou ser a forma mais adequada aos objetivos desta dissertação. A Seção 3.1 apresenta o processo utilizado para o mapeamento sistemático. O planejamento do estudo bem como a condução do trabalho e os resultados obtidos são apresentados e discutidos nas seções seguintes.

3.1 Processo do Mapeamento Sistemático

O processo utilizado para esse mapeamento sistemático pode ser sumarizado em três fases principais: `Planejamento, Execução` e `Análise dos resultados`. A Figura 3.1 sumariza essas fases. Na fase de `Planejamento`, estabelece-se o objetivo do mapeamento sistemático ressaltando a necessidade de realizá-lo. Nessa fase, o protocolo do mapeamento é produzido contendo os procedimentos básicos: formulação das questões de investigação em que o mapeamento será guiado; definição das *strings* de busca; seleção das bases de dados e descrição dos critérios de inclusão e exclusão. Na fase de `Execução`, as buscas são realizadas, os estudos encontrados são avaliados e classificados via critérios de inclusão e de exclusão e os dados são coletados. Em seguida, na `Análise dos resultados`, deve-se coletar e organizar os dados obtidos a fim de publicar os resultados [Biolchini et al., 2007].

Para apoiar esse mapeamento sistemático, foi utilizado neste trabalho o modelo de Biolchini et al. [2007] que pode ser definido em cinco etapas principais, descritas a seguir.

1. Formulação da pergunta: consiste em estabelecer o foco de interesse da pesquisa na formulação de perguntas de interesse, ou seja, o que se espera que seja esclarecido/respondido.

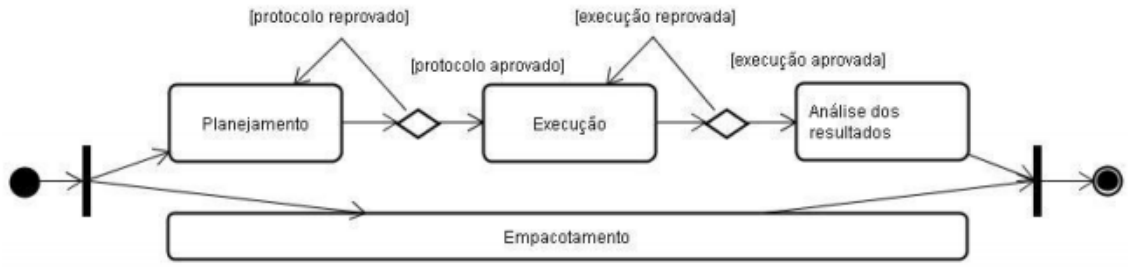

Figura 3.1. Processo de condução de estudo sistemático - Fonte: Biolchini et al. [2007]

2. Seleção de fontes: consiste em definir os repositórios de pesquisa e características como a linguagem dos estudos, *strings* de busca e métodos de pesquisa, entre outros.

3. Seleção de estudos: consiste em definir os critérios de inclusão e exclusão dos estudos, bem como realizar uma seleção dos trabalhos.

4. Extração de informação: consiste em definir os critérios de inclusão e exclusão das informações extraídas dos estudos selecionados, bem como estabelecer uma forma para exibir essas informações.

5. Sumarização de resultado: consiste em analisar e apresentar as informações extraídas.

3.2 Planejamento do Mapeamento Sistemático

Os estudos sistemáticos da literatura têm três objetivos principais [Kitchenham, 2004]:

- resumir a evidência existente em um conceito particular, por exemplo, um novo tratamento ou tecnologia;

- identificar eventuais lacunas na pesquisa atual, a fim de sugerir áreas para investigação; e

- fornecer uma estrutura ou um *background*, a fim de prestar um apoio adequado às novas atividades de investigação.

Neste contexto, o mapeamento sistemático realizado e descrito nesta dissertação teve como objetivo investigar como a literatura tem abordado a relação entre os valores referência de métricas de software orientados por objetos com falhas de softwares e

bad smells. Em particular, esse mapeamento sistemático se propõe a verificar se há evidências empíricas no que se refere a aplicação dos valores referência de métricas para predição de falhas e para a detecção de *bad smells*, pois caso não haja estudos com este foco, isso pode ser uma lacuna dos estudos atuais. Esses dois aspectos foram considerados por estarem fortemente relacionados à qualidade interna do produto de software.

Esse mapeamento sistemático visa: (i) coletar evidências de como a literatura tem aplicado valores referência de métricas de softwares orientados por objetos para predição de falhas, (ii) como a literatura tem investigado a correlação entre valores referência de métricas de software e *bad smells* e, (iii) fornecer um *background* e apoiar o presente trabalho.

3.2.1 Perguntas Definidas para o Mapeamento Sistemático

A fim de alcançar os objetivos do mapeamento sistemático foram estabelecidas as seguintes questões de pesquisa:

- ***QP1.*** *Como a literatura tem aplicado valores referência de métricas de softwares orientados por objeto para a detecção de bad smells?*

- ***QP2.*** *Como a literatura tem aplicado valores referência de métricas de softwares orientados por objeto para a predição de falhas em software?*

3.2.2 Seleção de Fontes de Pesquisa

O Portal da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES[1]) foi utilizado para a realização desse mapeamento sistemático. O Portal da CAPES foi escolhido porque é uma biblioteca virtual que possui um acervo de mais de 37 mil títulos com texto completo, 126 bases referenciais, 11 bases dedicadas exclusivamente a patentes, além de livros, enciclopédias e obras de referência, normas técnicas, estatísticas e conteúdo audiovisual.

Dentre os repositórios científicos acessados pela CAPES, destacam-se no contexto deste trabalho os seguintes: *Institute of Electrical and Electronic Engineers (IEEE), Association for the Computing Machinery (ACM), Computers & Applied Sciences Complete (CASC), Technology Research Database, World Scientific (WSP), Computer & Information Systems Abstracts*, dentre outros.

[1]http://www.periodicos.capes.gov.br

Além das questões de pesquisa, o protocolo de um mapeamento sistemático da literatura prevê a definição de palavras-chave, inerentes às questões formuladas, e seus sinônimos. As palavras-chave definidas e seus sinônimos utilizados nesse mapeamento sistemático são:

- métricas: *metrics*
- valor referência: *threshold, thresholds, reference value*
- predição de falhas: *fault prediction, failure prediction, bug prediction*
- *bad smell*: *bad smell, bad smells, code smell, code smells, design flaws, anti-patterns, design deviance, modularity anomalies*
- software: software

3.2.3 *Strings* de Busca

Foram considerados no mapeamento sistemático da literatura os estudos que apresentaram as palavras-chave relacionadas ou uma de suas variações. A partir das palavras-chave definidas, dos sinônimos e das restrições "AND" e "OR" previstas nos repositórios de pesquisa, foram construídas as seguintes *Strings* de Busca 1 e 2, que visam identificar estudos compatíveis com as questões de pesquisa *QP1* e *QP2* respectivamente.

1. *(METRIC OR METRICS) AND ("THRESHOLD" OR "THRESHOLDS" OR "REFERENCE VALUE") AND SOFTWARE AND ("BAD SMELL" OR "BAD SMELLS" OR "CODE SMELL" OR "CODE SMELLS" OR "DESIGN FLAW" OR "ANTI-PATTERNS" OR "DESIGN DEVIANCE" OR "MODULARITY ANOMALIES").*

2. *(METRIC OR METRICS) AND ("THRESHOLD" OR "THRESHOLDS" OR "REFERENCE VALUE") AND SOFTWARE AND ("FAULT PREDICTION" OR "FAILURE PREDICTION" OR "BUG PREDICTION").*

3.2.4 Critérios de Inclusão e Exclusão

Os critérios de seleção de estudo têm como objetivo identificar os estudos primários que fornecem evidências diretas sobre as questões de pesquisa. Nesse sentido, com base

nas questões de pesquisa apresentadas nessa dissertação, foram definidos os critérios de inclusão e exclusão dos estudos a seguir Kitchenham [2004].

- Critérios de Inclusão:
  - os documentos devem apresentar claramente uma relação entre os termos relacionados nas *strings* de busca;
  - os documentos devem estar completos, com no mínimo 4 páginas.
- Critérios de Exclusão:
  - documentos duplicados;
  - tutoriais, pôsteres, painéis, palestras, mesas redondas, oficinas.

3.3 Execução do Mapeamento Sistemático

Esta seção apresenta as etapas de seleção dos estudos e os resultados encontrados no mapeamento sistemático da literatura, considerando as duas *strings* de busca apresentadas na Seção 3.2.3. A execução das *strings* foi realizada em dezembro de 2015 e janeiro de 2016.

3.3.1 Seleção dos Estudos

Após a execução das *strings* de busca, as seguintes etapas de seleção de estudos foram utilizadas para aplicação dos critérios de inclusão e exclusão definidos na fase de planejamento:

1. leitura por título e resumo: leitura do título, do resumo e das palavras-chave dos trabalhos recuperados a fim de se obter os artigos relevantes, relacionados as questões de pesquisa, e excluir estudos que não sejam relevantes.

2. leitura diagonal: leitura da introdução, tópicos e conclusão dos artigos para verificar se o artigo realmente está relacionado às questões de pesquisa e assegurar que estudos que não são relevantes não foram selecionados. Caso o artigo não tenha relação direta às questões de pesquisa, ele é excluído do estudo.

3. leitura completa: leitura completa dos artigos selecionados para avaliação da relevância e extração dos dados.

Cada um desses passos requer a leitura de parte do trabalho a fim de identificar se ele é relevante para o mapeamento sistemático da literatura; caso haja alguma dúvida relacionada à relevância do estudo em uma etapa, ele é mantido para a próxima etapa da seleção dos estudos. A seleção dos estudos foi realizada por apenas um pesquisador.

3.3.2 Coleta dos Dados

O objetivo da *String* de Busca 1 foi encontrar trabalhos que relacionam valor referência de métricas orientadas por objeto e *bad smells*. Ela retornou dois resultados que são apresentados na Tabela 3.1.

Tabela 3.1. Resultados relevantes da *String* de Busca 1

#Trabalho	Ano	Título	Autor	Citações
P1	2014	Code-smell detection as a bilevel problem	Dilan Sahin, Marouane Kessentini Slim Bechikh	-
P2	2014	Object oriented software metrics threshold values at quantitative acceptable risk levek	Satwinder Singh K. S. Kahlon	-

O objetivo da *String* de Busca 2 foi encontrar trabalhos que relacionam valor referência de métricas orientadas por objeto à predição de falhas. Ela retornou dez resultados porém, dois dos resultados foram desconsiderados na Etapa 4 do processo de seleção dos estudos, leitura completa, por não utilizar valores referência de métricas de software orientados por objeto como foco do estudo. Os resultados considerados como válidos para esta *string* são reportados na Tabela 3.2.

3.3.3 Extração de Informação

O processo de extração dos dados foi realizado durante a Etapa 4 da seleção dos estudos. Nessa etapa, além dos dados coletadas para análise de relevância dos trabalhos, foram coletadas as informações a seguir.

- Dados Gerais: título do trabalho, autoria, ano, local da publicação, área de conhecimento do veículo de publicação do trabalho, palavras-chave, número de citações.

- Dados de Pesquisa: objetivo de se utilizar valores referência no trabalho, predição de falhas e/ou identificação de *bad smells*, quantidade de softwares verificados, tipos de softwares verificados, linguagem de programação dos softwares verificados, *bad smells* analisados e métricas utilizadas.

Tabela 3.2. Resultados relevantes da *String* de Busca 2

#Trabalho	Ano	Título	Autor	Citações
P3	2010	Thresholds based outlier detection approach for mining,class outliers: An empirical case study,on software measurement datasets	Oral Alan Cagatay Catal	6
P4	2011	Can traditional fault prediction models be used for vulnerability prediction?	Yonghee Shin Laurie Williams	5
P5	2011	Class noise detection based on software metrics and ROC curves	Cagatay Catal Oral Alan Kerime Balkan	9
P6	2013	An empirical study of the effect of power Law distribution,on the interpretation of OO metrics	Raed Shatnawi Qutaibah Althebyan	-
P7	2013	A study of subgroup discovery approaches for defect prediction	Daniel Rodriguez Roberto Ruiz Jose C. Riquelme Rachel Harrison	4
P8	2014	An empirical study based on semi-supervised hybrid self-organizing map for software fault prediction	Golnoush Abaei, Ali Selamat Hamido Fujita	2
P9	2015	Fault prediction considering threshold effects of object-oriented metrics	Ruchika Malhotra Ankita Jain Bansal	1
P10	2015	Deriving metrics thresholds using log transformation	Raed Shatnawi	25

3.3.4 Análise dos Resultados

Nesta seção são apresentados os resultados encontrados no mapeamento sistemático para as *strings* de busca executadas.

3.3.4.1 *QP1* - Valores Referência X *Bad Smells*

Uma breve descrição é apresentada a seguir para cada um dos trabalhos recuperados pela *String* de Busca 1 referente a *QP1* desse mapeamento sistemático cujo objetivo é identificar como a literatura tem investigado a aplicação de valores referência de métricas de softwares orientados por objeto à detecção de *bad smells*.

Sahin et al. [2014]

Esses autores afirmam que *code smells* são detectados, de um modo geral, por meio de métricas de qualidade que representam alguns sintomas dos *code smells*. No entanto, a seleção do conjunto de métricas de qualidade é um desafio devido à ausência de consenso acerca da estratégia de identificação de alguns *code smells* baseados em um conjunto de sintomas, e também devido ao elevado esforço em determinar manualmente o valor referência para cada métrica. Sahin et al. [2014] propõem tratar a geração de regras para detecção de *code smell* como um problema de otimização de dois níveis, no qual o problema de nível superior gera um conjunto de regras de detecção, uma combinação

de métricas de qualidade, o que maximiza a cobertura da base de exemplos de *code smells* e o problema de nível mais baixo gera *code smells* artificiais. O nível mais baixo maximiza o número de *code smells* artificiais gerados que não podem ser detectados pelas regras produzidas pelo nível superior.

De acordo com Sahin et al. [2014], a principal vantagem da formulação dos dois níveis propostos é que a geração de regras de detecção não se limita a alguns exemplos de *code smells* identificados manualmente por desenvolvedores, mas permite a predição do comportamento de novos *code smells* diferentes daqueles da base de exemplos. A proposta foi avaliada em mais de 31 execuções em nove sistemas de código aberto e um projeto industrial. Os resultados da avaliação mostraram que sete tipos de *code smells* foram detectados com uma média de mais de 86% em termos de precisão e *recall*. Sahin et al. [2014] consideram que o desempenho da proposta de dois níveis é superior aos de outras técnicas de detecção de *code smells* disponíveis na literatura.

Singh & Kahlon [2014]

O trabalho de Singh & Kahlon [2014] é baseado na ideia de que a propensão de um módulo ao erro está de alguma forma associada com um projeto ruim. Assim, valores referência de métricas associados com *bad smells* podem ser utilizados para prever falhas. Singh & Kahlon [2014] estudam os valores referência das métricas contra *bad smells* utilizando análise de risco em cinco níveis diferentes. Três versões do *Mozilla Firefox* foram utilizadas como conjunto de dados no estudo. Os resultados mostraram que algumas métricas têm valores referência para vários níveis de risco que são de uso prático em prever classes defeituosas. Um dos valores referência foi selecionado entre os diversos níveis de risco, por *bad smell*, determinando a maior área sob a curva ROC *(Receiver Operating Character)* para as classes defeituosas nos níveis de risco correspondente.

Os valores referência derivados no trabalho Singh & Kahlon [2014], foram validados por meio da identificação de classes defeituosas para versões do *Firefox* subsequentes. Os resultados obtidos nos experimentos foram positivos, mas para validar a pesquisa, de acordo com os autores, mais experimentos precisam ser realizados com diferentes linguagens de programação orientadas por objeto, bem como em aplicações industriais.

3.3.4.2 *QP2*- Valores Referência X Predição de falhas

A seguir, é apresentada uma breve descrição de cada um dos trabalhos recuperados pela *string* de busca referente a *QP2* deste mapeamento sistemático cujo objetivo é

identificar como a literatura tem aplicado valores referência à predição de falhas.

Alan & Catal [2011]

Esses autores propõem uma abordagem de detecção de classes *outlier*. *Outlier* é aquela classe que parece desviar-se acentuadamente dos outros integrantes da amostra em que ela ocorre. A abordagem usa valores referência de métricas de software e rótulos (*labels*) de classes para identificar as classes que são *outlier*. Os autores avaliaram essa proposta em um conjunto de dados públicos da NASA disponível no repositório *Promise* e utilizaram sete métricas e seus valores referência conhecidos de pesquisas empíricas. As métricas utilizadas foram: *Lines of code (LoC), Cyclomatic complexity (CC), Essential complexity (EsC), Unique operator (UOp), Unique operand (UOpnd), Total operator (TOp) e Total operand (TOpnd)*. Antes de construir o método de detecção de *outlier*, os autores examinaram as características de cada módulo do conjunto de dados. Eles perceberam que existiam vários módulos com valores muito baixos de métricas mas com rótulos de classe defeituosa e alguns módulos com valores muito elevados de métricas, mas com rótulos de classe não-defeituosa.

A partir dessa observação, Alan & Catal [2011] definiram as seguintes questões de pesquisa que conduziram o trabalho: valores referência de métricas de software podem auxiliar na identificação de módulos defeituosos que são registrados como não-defeituoso em um conjunto de dados? Valores referência de métricas de software podem auxiliar a identificar módulos não-defeituosos que são registrados como defeituosos em um conjunto de dados? A abordagem de detecção de *outlier* baseada em valores referência de métricas pode melhorar o desempenho de algoritmos de predição de falhas tais como *Naive Bayes* e *Random Forests*? De acordo com os resultados dos experimentos realizados, Alan & Catal [2011] chegaram a conclusão de que a abordagem de detecção de *outlier* baseada em valores referência de métricas é uma abordagem eficaz e eficiente para detectar classes *outlier* no campo da Engenharia de Software.

Shin & Williams [2011]

Shin & Williams [2011] afirmam que falhas e vulnerabilidades têm pontos em comum que podem permitir que as equipes de desenvolvimento utilizem modelos tradicionais de predição de falhas e métricas para a previsão de vulnerabilidades. De acordo com o trabalho, um modelo de previsão de falha é um modelo utilizado para prever a probabilidade de que um arquivo terá, pelo menos, uma falha. E um modelo de previsão de vulnerabilidade é um modelo usado para prever a probabilidade de que um arquivo terá pelo menos uma vulnerabilidade. No trabalho, falha é considerada uma condição

acidental que faz com que uma unidade funcional falhe ao realizar a sua função. Vulnerabilidade é definida como a instância de um erro na especificação, no desenvolvimento ou na configuração do software de tal forma que, a sua execução, pode violar a política de segurança Shin & Williams [2011].

Shin & Williams [2011] realizaram um estudo empírico para determinar se modelos de previsão de falhas podem ser usados para predição de vulnerabilidades, ou se modelos de predição de vulnerabilidades especializados devem ser desenvolvidos quando ambos os modelos são construídos com métricas tradicionais de complexidade, métricas referentes à código *churn* e métricas de histórico de falhas. O estudo foi realizado utilizando o navegador *Mozilla Firefox*, no qual 21% dos arquivos de código-fonte tem falhas e apenas 3% dos arquivos têm vulnerabilidades. Tanto o modelo de predição de falhas quanto o modelo de predição de vulnerabilidade forneceram capacidade semelhante na predição de vulnerabilidade em uma ampla gama de classificação de valores referência das métricas utilizadas. Os resultados sugerem que modelos de predição de falhas com base em métricas tradicionais podem substituir modelos especializados de predição de vulnerabilidades. Em relação à contribuição do estudo de Shin et al. Shin & Williams [2011] no que concerne a valores referência de métricas de software, pode-se destacar que a análise desses valores mostra que a previsão de vulnerabilidade pode exibir uma diminuição drástica no *recall* com um aumento de precisão para determinado intervalo de classificação de valores referência.

Catal et al. [2011]

Esses autores propõem um algoritmo de detecção de ruído (*noise*) baseado em valores referência de métricas de software. Os valores referência são obtidos a partir da Análise ROC *(Receiver Operating Characteristic)*. Catal et al. [2011] realizaram estudos de caso em cinco bancos de dados públicos da NASA e detalharam a construção de modelos de previsão de falhas baseados em *Naive Bayes*, tanto antes como depois da aplicação do algoritmo de detecção de ruído proposto. Os resultados experimentais mostraram que esta abordagem de detecção de ruído é muito eficaz para a detecção da classe de ruído e que o desempenho dos preditores de falhas utilizando um algoritmo *Naive Bayes* com um filtro *logNum* melhora se os rótulos das classes de módulos identificados como ruidosos são corrigidos. A principal contribuição do trabalho de Catal et al. [2011] é a proposta de uma técnica de detecção de ruído com base nos valores referência de métricas de software que estende uma técnica de cálculo de valor referência de métricas para proporcionar um melhor processo de tomada de decisão.

Shatnawi & Althebyan [2013]

Shatnawi & Althebyan [2013] afirmam que as métricas de software podem ser usadas para encontrar possíveis problemas ou oportunidades para melhoria da qualidade mas, que no entanto, os valores referência das métricas são números difíceis de interpretar. O objetivo do trabalho deles foi validar o efeito da lei de potência sobre a interpretação das métricas de software. Eles estudaram cinco sistemas de código aberto para investigar a distribuição e os seus efeitos sobre os modelos de predição de falhas. Os resultados obtidos mostraram que o comportamento da lei de potência tem um efeito sobre a interpretação e uso de métricas de software, em particular, no caso das métricas CK. Os valores referência são derivados das propriedades da distribuição da lei de potência quando aplicados a sistemas de código aberto. De acordo com Shatnawi & Althebyan [2013], as propriedades de uma distribuição de lei de potência pode ser eficaz para melhorar os modelos de predição de falhas definindo valores referência razoáveis.

Rodriguez et al. [2013]

Os autores sugerem o uso de uma abordagem descritiva para a predição de falhas em vez de técnicas de classificação precisas que são normalmente adotadas. Isso permite caracterizar módulos defeituosos com regras simples que podem ser facilmente aplicadas por profissionais e oferecer uma abordagem prática (ou de engenharia) ao invés de um resultado altamente preciso. Os autores descrevem dois algoritmos de descoberta de subgrupos, o algoritmo SD e o algoritmo CN2-SD para obter as regras que identificam os módulos propensos a defeitos. A avaliação empírica da proposta é realizada em um conjunto de dados público disponível no repositório *"Promise"*. Também foram utilizadas métricas orientadas por objeto relacionadas com predição de defeitos. Os resultados mostraram que as regras geradas podem ser usadas para orientar o esforço de teste, a fim de melhorar a qualidade dos projetos de desenvolvimento de software. Essas regras podem indicar métricas, seus valores referência e o relacionamento entre as métricas de módulos defeituosos. De acordo com Rodriguez et al. [2013], as regras induzidas são simples de usar e fáceis de compreender uma vez que fornecem uma descrição em vez de uma classificação completa de todo o conjunto de dados.

Abaei et al. [2015]

Abaei et al. [2015], propõem um modelo de detecção de falhas de software automatizado usando mapa auto-organizado híbrido semi-supervisionado *(HySOM)* para detectar módulos com propensão a falhas em projetos de software com alta precisão e

melhorar a capacidade de modelos de detecção generalizados. *HySOM* é um modelo semi-supervisionado com base no mapa de auto-organização e rede neural artificial. De acordo com Abaei et al. [2015] a vantagem desse modelo é a capacidade de prever o rótulo dos módulos em uma forma semi-orientada usando valores referência de medição de software na ausência de dados sobre a qualidade.

O modelo de Abaei et al. [2015] foi testado em oito softwares, sendo eles conjuntos de dados públicos e dados industriais. Os resultados mostraram uma melhoria na taxa de falsos negativos e na taxa de erro global em 80% e 60% dos casos, respectivamente, para os conjuntos de dados. Além disso, foi investigado o desempenho do modelo proposto em comparação a outros métodos propostos na literatura tais como *Naive Bayes, Random Forest* e redes neurais artificiais. Abaei et al. [2015] consideram que, de acordo com os resultados, o modelo semi-supervisionado pode ser usado pelos gerentes de projetos, desenvolvedores de software e testadores como uma ferramenta automatizada para orientar o esforço, auxiliando a priorizar os módulos mais propensos a falhas, proporcionando melhoria da qualidade do desenvolvimento de software e obtendo testes em menos tempo e com menos custo.

Malhotra & Bansal [2015]

O estudo de Malhotra & Bansal [2015] investiga empiricamente a relação entre métricas orientadas por objeto e propensão a falhas. O trabalho descreve um estudo para construir modelos preditivos para identificar partes do software que têm alta probabilidade de ocorrência de falhas. Os autores consideraram o efeito de valores referência de métricas orientadas por objetos sobre propensão a falhas e construíram modelos preditivos com base nos valores referência das métricas utilizadas. Utilizou-se um modelo estatístico derivado de regressão logística para calcular os valores referência das métricas orientadas por objeto de Chidamber & Kemerer [1994].

Malhotra & Bansal [2015] demonstraram os efeitos dos valores referência em diferentes níveis de risco (baixo risco e alto risco) e validaram o uso desses valores referência em seis softwares públicos e proprietários, usando vários métodos de aprendizado de máquina e classificadores de mineração de dados. A partir dos resultados, Malhotra & Bansal [2015] concluíram que a metodologia de valores referência proposta funciona bem para os projetos de natureza semelhante ou de características semelhantes. Eles concluíram também que os valores referência ajudam na detecção precoce das classes propensas a falha, logo que os valores de métricas orientadas por objeto estão disponíveis. Assim, as classes que são previstas como propensas a falha podem ser redesenhadas ou modificadas de modo a torná-las parte da categoria de baixo-risco.

Shatnawi [2015]
Shatnawi [2015] propõe uma metodologia baseada na transformação logarítmica para melhorar as métricas de qualidade. O autor utilizou a transformação logarítmica para reduzir o efeito de assimetria em dados. Para explorar o efeito da transformação logarítmica na análise dos dados, foi realizada uma análise da utilização de métricas de software após a identificação de áreas propensas a falhas em várias versões de 11 projetos de código aberto Java diferentes, totalizando 41 *releases*.

De acordo com Shatnawi [2015], os resultados mostraram que a transformação logarítmica pode ser usada para derivar valores referência para todas as métricas sob investigação. Os resultados da transformação foram usados para conduzir a classificação de propensão a falhas baseado nos valores referência comparando-a com os resultados sem transformação. A classificação de falhas com a transformação foi mais bem sucedida do que a sem transformação. A comparação estatística mostrou melhor classificação de anomalia utilizando dados transformados em logaritmos do que o contrário. Sendo assim, o autor sugere utilizar a transformação logarítmica antes de avaliar a qualidade do software.

3.4 Discussão dos Resultados

Poucos trabalhos encontrados nesse mapeamento sistemático da literatura discutem a utilização de valores referência na indústria de software, provavelmente devido à falta de ferramentas e, principalmente, a ausência de valores referência que possam orientar o seu uso em relação à avaliação da qualidade de produto de software.

Observa-se por meio dos dados relacionados nas Tabelas 3.3 e 3.4 que são utilizados um número restrito de métricas nos estudos realizados. A maior parte dos trabalhos utilizam como objeto de pesquisa apenas as métricas CK de Chidamber & Kemerer [1994] o que faz com que exista uma lacuna de pesquisa, pois existem várias métricas na literatura além das métricas CK que poderiam ser avaliadas por meio de valores referência. Também é possível observar que os estudos foram realizados em uma quantidade limitada de sistemas nos quais as linguagens de programação dominantes são C, C++ e Java.

Há mais trabalhos investigando a relação de valores referência de métricas com falhas do que com *bad smells*, 9 em um total de 10 trabalhos retornados pelas *strings*

de busca. Nos trabalhos que investigam a relação com *bad smells* foram estudados 13 *bad smells*. Apenas um dos trabalhos estuda os dois aspectos: *bad smells* e predição de falhas, relacionando-os com valores referência. Todos os trabalhos foram publicados entre 2010 e 2015, o que mostra que atualmente a comunidade acadêmica tem se interessado por esse assunto.

Tabela 3.3. Sumarização dos resultados

#	Autor(es)	Aplicação Valor Referência	Softwares Avaliados			*Bad Smells* Analisados	Métricas Utilizadas
			Qtde	Tipo	Linguagens		
P1	-Dilan Sahin, -M. Kessentini -Slim Bechikh	Detecção de *Bad Smells*	10	Software Livre e Software Proprietário	C e C++	-Blob -Feature Envy -Data Class -Spaghetti Code -Functional Decomposition -Lazy Class -Long Parameter List	-Weighted Methods per Class (WMC) -Response for a Class (RFC) -Lack of Cohesion of Methods (LCOM) -Number of Attributes (NA) -Attribute Hiding Factor (AH) -Method Hiding Factor (MH) -Number of Lines of Code (NLC) -Coupling Between Object classes (CBO) -Number of Association (NAS) -Number of Classes (NC) -Depth of Inheritance Tree (DIT) -Polymorphism Factor (PF) -Attribute Inheritance Factor (AIF) -Number of Children (NOC)
P2	-S. Singh -K. S. Kahlon	Detecção de *Bad Smells* e Predição de Falhas	3	Software Livre	C e C++	-Large Method -LongParameter List -Large Class -Temporary Fields -ShotgunSurgery -Lazy Class -Data Class -Speculative Generality -Middle Man -Feature Envy Inappropriate Intimacy	-No. of Children (NOC) -Depth of Inheritance Tree (DIT) -Coupling Between Objects (CBO) -Response For Class (RFC) -Weighted Method Count (WMC) -Lack of Cohesion in Methods (LCOM) -Number of Attributes (NOA) -Public,Factor,(PuF): -The Encapsulation Factor (EncF) -Number of Abstract Methods (NOAM) -Number of Overridden Methods (NOOM) -Cohesion (Co)
P3	-Oral Alan -Cagatay	Predição de Falhas	5	Software Livre	C e C++	Não se aplica.	-Lines of code (LoC) -Cyclomatic complexity (CC) -Essential complexity (EsC) -Unique operator (UOp) -Unique operand (UOpnd) -Total operator (TOp) -Total operand (TOpnd)
P4	-Yonghee Shin -Laurie Williams	Predição de Falhas	1	Software Livre	C e C++	Não se aplica.	-Lines of code (LoC) -CountLineCodeDecl -CountDeclFunctionEssential -EssentialComplexity -CyclomaticStrict -MaxNesting -CommentDensity -FanIn -FanOut -NumChanges -LinesChanged -LinesInserted -LinesDeleted -LinesNew -NumPriorFaults
P5	-Cagatay Catal -Oral Alan -Kerime Balkan	Predição de Falhas	5	Software Livre	C e C++	Não se aplica.	-McCabe?s lines of code -Cyclomatic Complexity -Essential Complexity -Design Complexity -Halstead?s line count -Count of lines of comments -Count of blank lines -Count of lines of code and comment -Unique Operator -Unique Operand -Total Operator -Total Operand -Branch Count

Tabela 3.4. Sumarização dos resultados - continuação

#	Autor(es)	Aplicação Valor Referência	Softwares Avaliados			*Bad Smells* Analisados	Métricas Utilizadas
			Qtde	Tipo	Linguagens		
P6	-Raed Shatnawi -Qutaibah,Althebyan	Predição de Falhas	5	Software Livre	Java	Não se aplica.	-Coupling between objects (CBO) -Depth of inheritance tree (DIT) -Number of children (NOC) -Response set for a class (RFC) -Weighted methods complexity (WMC) -Source lines of code (SLOC) -Number of variables (NOVs) -Number of methods (NOMs)
P7	-Daniel Rodriguez -Roberto Ruiz -Jose Riquelme -Rachel Harrison	Predição de Falhas	13	Software Livre e Software Proprietário	C, C++ e Java	Não se aplica.	-McCabe?s Lines of code -Cyclomatic complexity -Essential complexity -Design complexity -Unique operators -Unique operands -Total operators -Total operands -No. of branches of the flow graph -Does the module contain defects? -Weighted Method Count (WMC) -Depth of Inheritance Tree (DIT) -Coupling Between Objects (CBO) -No. of Children (NOC) -Lack of Cohesion in Methods (LCOM) -Response For Class (RFC)
P8	-G. Abaei -Ali Selamat -Hamido Fujita	Predição de Falhas	8	Software Livre e Software Proprietário	C e C++	Não se aplica.	-Lines of code (LoC) -Cyclomatic complexity (CC) -Unique operator (UOp) -Unique operand (UOpnd) -Total operator (TOp) -Total operand (TOpnd)
P9	-R. Malhotra -Ankita Bansal	Predição de Falhas	6	Software Livre e Software Proprietário	C++ e Java	Não se aplica.	-Weighted Method Count (WMC) -Depth of Inheritance Tree (DIT) -No. of Children (NOC) -Coupling Between Objects (CBO) -Response For Class (RFC) -Lack of Cohesion in Methods (LCOM) -Lines of code (LOC)
P10	-R. Shatnawi	Predição de falhas	11	Software Livre	Java	Não se aplica.	-Weighted Method Count (WMC) -Depth of Inheritance Tree (DIT) -No. of Children (NOC) -Coupling Between Objects (CBO) -Response For Class (RFC) -Lack of Cohesion in Methods (LCOM)

A partir das análises realizadas nos artigos obtidos, as respostas para as questões de pesquisa definidas para este mapeamento sistemático da literatura, são as que seguem:

QP1. *Como a literatura tem aplicado valores referência de métricas de softwares orientados por objeto para a detecção de bad smells?*

Por meio dos trabalhos retornados nas *strings* de busca, pode-se perceber que na literatura, os pesquisadores têm proposto um conjunto de regras, que são combinações de métricas de softwares com os seus valores referência, para a detecção de *bad smells* em um sistema. A análise estatística dos resultados obtidos nos sistemas que tiveram as regras aplicadas em conjunto com outras técnicas como algoritmo de otimização,

demonstraram melhor *Recall* e Precisão [Wikipédia, 2016] em comparação a outros métodos de pesquisa. Outra abordagem encontrada na literatura foi a derivação de valores referência de métricas de software para identificar *bad smells* usando análise de riscos. Alguns pesquisadores também propõem a relação de que os valores referência derivados para detectar *bad smell* podem ser verificados por meio de predição de classes defeituosas.

QP2. *Como a literatura tem aplicado valores referência de métricas de softwares orientados por objeto para a predição de falhas em software?*

Observa-se por meio dos resultados obtidos pelas *strings* de busca que, alguns trabalhos relacionados a predição de falhas utilizam algoritmos de aprendizado de máquina tais como *Naive Bayes e Random Forests.* Algumas abordagens também têm sido propostas envolvendo valores referência e *labels* (rótulos) de classes para identificar valores atípicos *(outiliers)* para realizar a predição de falhas. Outros trabalhos, sugerem modelos que utilizam a classificação de valores referência para identificar vulnerabilidades além de falhas. Outros estudos propõem algoritmos de predição de falhas que utilizam valores referência derivados por meio de análise *ROC (Receiver Operating Characteristic).* Além disso, alguns autores demonstraram que distribuição de lei de potência pode ser eficaz em melhorar modelos de predição de falhas. Também encontram-se na literatura metodologias baseadas em regressão logística e validação de *interproject* utilizadas em conjunto com valores referência para realizar a predição de falhas.

3.5 Ameaças à Validade

Considera-se como principal ameaça à validade desse mapeamento sistemático da literatura, a estratégia de busca utilizada. Visto que foi utilizado engenhos de busca automatizados, estudos relevantes podem não ter sido retornados nos resultados. Apesar de identificadas as palavras-chaves relevantes para compor as *strings* de busca, algum estudo pode não ter sido encontrado por utilizar um termo diferente do previsto.

Outro fator que pode ser considerado como uma ameaça à validade é a pequena quantidade de estudos retornados. Mesmo com as buscas automatizadas, sem qualquer restrição de período, apenas 10 estudos foram selecionados. Outras fontes de dados poderiam ter sido adicionadas à pesquisa a fim de obter mais resultados.

Por fim, de acordo com Kitchenham & Charters [2007], o fato do processo de busca e de seleção dos estudos primários terem sido realizados por apenas um pesquisador, que possui conhecimento básico via literatura sobre *bad smells* e predição de falhas, configura uma limitação do estudo.

3.6 Considerações Finais

Este mapeamento sistemático da literatura foi realizado com o objetivo de coletar e analisar as publicações que utilizam valores referência de métricas de softwares orientados por objeto para predição de falhas e identificação de *bad smells*. Foram retornados trabalhos publicados no período de 2010 a 2015 disponíveis nos repositórios de publicações científicas relevantes, tais como *CAPES, ACM Digital Library, IEEE Xplore, Science Direct e Scopus*. Isso demonstra que esse é um assunto que está em discussão atualmente.

Os resultados desse mapeamento sistemático sugerem que a aplicação de valores referência é considerada eficaz para indicar a qualidade dos softwares, provendo um referencial para a avaliação quantitativa da qualidade interna de softwares orientados por objeto e contribuindo para a predição de falhas e detecção de *bad smells*. Porém, observa-se a necessidade de estudos que objetivem avaliar um número maior de métricas de softwares orientados por objetos, bem como avaliar a utilidade dos valores referência já propostos na literatura na predição de falhas e na identificação de *bad smells*. Essas observações indicam o cenário em que o presente trabalho se situa e enfatiza a contribuição almejada com os seus resultados.

Capítulo 4

Metodologia

Este capítulo descreve o processo que envolve a preparação e condução dos experimentos realizados para responder as seguintes questões de pesquisa investigadas neste trabalho:

QP1. *Os valores referência de métricas de software orientados por objetos auxiliam a identificar bad smells?*

Com essa questão de pesquisa genérica, pretende-se verificar se os valores referência podem ser úteis na detecção de *bad smells*. Essa questão genérica é dividida em duas mais específicas, conforme descrito a seguir.

QP1.1. *Qual é a eficácia da detecção de bad smells utilizando-se estratégias baseadas nos valores referência e tomando-se como base os resultados gerados por ferramentas de detecção de bad smells*?

O objetivo da análise realizada para responder essa questão de pesquisa é colocar os resultados das estratégias de detecção propostas nesta dissertação em perspectiva em relação aos resultados de outras formas de detecção disponíveis. Duas ferramentas foram utilizadas neste estudo: *JSpIRIT* e *JDeodorant*. O experimento realizado para responder essa questão de pesquisa é apresentado no Capítulo 6.

QP1.2. *Os valores referência apoiam efetivamente a detecção de bad smells em relação a listas de referência geradas por um especialista com conhecimentos em orientação por objetos e bad smells?*

Na análise referente a *QP1.1*, ferramentas para detecção automatizada de *bad smells* foram utilizadas. Porém a eficácia dessas ferramentas não é conhecida e observam-se diferenças entre os resultados gerados por elas. Para uma melhor

avaliação das estratégias de detecção propostas neste trabalho, um especialista com conhecimentos em programação orientada por objetos (POO) e *bad smells* foi convidado a compor e validar listas de referência. O experimento realizado para responder essa questão de pesquisa é apresentado no Capítulo 7.

QP2. *Os valores referência de métricas de software orientados por objetos auxiliam a predizer falhas em um software?*

Por meio dessa questão de pesquisa pretende-se verificar a utilidade dos valores referência em predizer falhas de software. O experimento realizado para responder essa questão de pesquisa é apresentado no Capítulo 8.

A presente dissertação adotou uma metodologia experimental apoiada no referencial teórico da área. A partir da realização de um mapeamento sistemático da literatura e leitura das referências bibliográficas levantadas a cerca de trabalhos realizados, foram delineadas as etapas descritas nas próximas seções.

4.1 Definição dos Experimentos

Para responder as questões de pesquisa QP1 e QP2 foram definidos dois conjuntos de experimentos:

O primeiro conjunto de experimentos tem o objetivo de avaliar a relação de valores referência de métricas de software e *bad smells*. Ele visa responder a *QP1* e, para isso, foi dividido em duas partes:

1. Execução de experimentos com ferramentas de detecção de *bad smells*: o objetivo desse experimento é responder a *QP1.1*. Para isso é realizada uma comparação dos resultados obtidos pelas estratégias de detecção propostas nessa dissertação com os resultados obtidos por ferramentas de detecção de *bad smells*.

   Foi realizado um levantamento das ferramentas de identificação de *bad smell* disponíveis com base no trabalho de Fernandes et al. [2016]. O resultado desse levantamento identificou 12 ferramentas: *True Refactor, Code Bad Smell Detector, InCode, InFusion, IntelliJ IDEA, Stench Blossom, Gendarme, Checkstyle, PMD iPlasma, JSpirit, Jdeodorant*. Essas ferramentas foram avaliadas para se verificar a viabilidade de utilização delas no estudo. Na utilização de algumas delas, foram observados os seguintes problemas principais: (i) ferramentas não disponíveis para *download*, (ii) necessidade de licença para obter a acesso a todas

as funcionalidades da ferramenta (iii) ausência de funcionalidade para exportação dos resultados. Por essa razão, apenas duas das ferramentas analisadas se mostraram adequadas ao uso nesse trabalho: *JSpIRIT e JDeodorant*.

As ferramentas *JSpIRIT e JDeodorant* estão disponíveis para *download* e estão em constante desenvolvimento e manutenção. Além disso, os resultados obtidos por essas ferramentas podem ser exportados e são de fácil entendimento.

Uma inspeção manual também é realizada nesse experimento para averiguação dos resultados. Ressalta-se que os resultados das ferramentas de detecção não são tratados nesta dissertação como "oráculos", ou seja, verdadeiros, mas servem como lista de referência para colocar em perspectiva as estratégias de detecção definidas nesta dissertação, que são baseadas em valores referência de métricas de software.

2. Execução de experimentos com especialista: o objetivo dessa etapa é responder a *QP1.2*. Para tal, nesse experimento utilizam-se listas de referência geradas por um especialista que possui conhecimentos em programação orientada por objeto e que conhece a definição dos *bad smells* de Fowler [1999].

O segundo conjunto de experimentos foi realizado com o objetivo de identificar a relação entre valores referência de métricas de software e predição de falhas, respondendo a *QP2*. Nesse experimento são analisados dados de falhas de 12 sistemas de software *Java* abertos. Tem-se por objetivo, aqui, averiguar a existência de relação entre os valores referência e a ocorrência de falhas em software.

Os conjuntos de experimentos e seus resultados são detalhados nos Capítulos 6, 7 e 8 desta dissertação.

4.2 Seleção de Sistemas de Software

O *Qualitas Corpus* foi o repositório de sistemas definido para ser utilizado nos experimentos realizados nessa dissertação. O *Qualitas.class* é uma versão compilável do *Qualitas Corpus* proposto por Tempero et al. [2010], disponibilizada por Terra et al. [2013] para a comunidade científica. Nesse trabalho foram usados os dados do *Qualitas.class*.

O *Qualitas Corpus* foi escolhido como *dataset* para os experimentos porque possui uma ampla coleção de softwares de código aberto desenvolvidos em Java com a finalidade de serem utilizados em estudos empíricos de artefatos de códigos.

Segundo Tempero et al. [2010], o objetivo principal desse repositório é prover um recurso para que estudos acerca de softwares possam ser reproduzidos. As versões compiladas dos projetos Java na ferramenta *Eclipse* incluem os 111 sistemas do *Corpus*. O conjunto de softwares contém mais de 18 milhões de linhas de código, 200.000 classes e 1.5 milhões de métodos compilados [Filó, 2014].

Os estudos realizados nesse trabalho envolvem a realização de experimentos que demandam muito tempo. Por essa razão, dentre outros motivos, não foram analisados todos os sistemas do *Qualitas.class*, mas foram utilizados 12 sistemas do *Qualitas.class*, selecionados aleatoriamente para o estudo com *bad smells*: *Aoi, Checkstyle, Cobertura, Displaytag, Itext, Jedit, Joggplayer, jsXe, PMD, Quartz, Squirrel, Webmail*. Para a análise que envolve coleta de dados de falhas, foram analisados 10 sistemas do *Qualitas.class*, definidos em função da disponibilidade dos dados históricos de falhas dos sistemas: *Ant, AspectJ, Batik, Cayenne, Derby, JMeter, Maven, MyFaces, Poi, Roller*.

4.3 Seleção de Catálogo de Valores Referência

Para a composição de estratégias para detectar *bad smells*, foi selecionado o catálogo de valores referência proposto por Filó et al. [2015]. Foram consideradas três razões para a utilização desse catálogo: (i) os valores referência propostos por Filó et al. [2015] consideram a frequência de vários softwares e não simplesmente a média dos valores, (ii) os valores referência foram avaliados empiricamente e (iii) este catalogo é o que possui o maior número de valores referência propostos dentre os trabalhos encontrados na literatura. A Figura 4.1 apresenta o catálogo de valores referência proposto por Filó et al. [2015].

4.4 Seleção de Catálogo de *Bad Smells*

Para esse estudo selecionou-se o catálogo de *bad smells* de Fowler [1999] visto que é um dos mais utilizados na literatura. Fowler [1999] define o termo *bad smells* como um indicador de possível problema estrutural em código-fonte, que pode ser melhorado via refatoração e apresenta uma lista de *bad smells* abordando refatoração em código-fonte para cada um deles. Fowler [1999] não utiliza ou propõe estratégias de detecção para identificar os *bad smells* mas cita características de como reconhecer as anomalias no código-fonte.

Métrica	Bom/Frequente	Regular/Ocasional	Ruim/Raro
CA	$CA \leq 7$	$7 < CA \leq 39$	$CA > 39$
CE	$CE \leq 6$	$6 < CE \leq 16$	$CE > 16$
MLOC	$MLOC \leq 10$	$10 < MLOC \leq 30$	$MLOC > 30$
NOC	$NOC \leq 11$	$11 < NOC \leq 28$	$NOC > 28$
NOF	$NOF \leq 3$	$3 < NOF \leq 8$	$NOF > 8$
NOM	$NOM \leq 8$	$8 < NOM \leq 14$	$NOM > 14$
NORM	$NORM \leq 2$	$2 < NORM \leq 4$	$NORM > 4$
NSC	$NSC \leq 1$	$1 < NSC \leq 3$	$NSC > 3$
NSF	$NSF \leq 1$	$1 < NSF \leq 5$	$NSF > 5$
NSM	$NSM \leq 1$	$1 < NSM \leq 3$	$NSM > 3$
PAR	$PAR \leq 2$	$2 < PAR \leq 4$	$PAR > 4$
SIX	$SIX \leq 0,019$	$0,019 < SIX \leq 1,333$	$SIX > 1,333$
VG	$VG \leq 2$	$29 < VG \leq 4$	$VG > 4$
WMC	$WMC \leq 11$	$11 < WMC \leq 34$	$WMC > 34$
DIT	$DIT \leq 2$	$2 < DIT \leq 4$	$DIT > 4$
LCOM	$LCOM \leq 0,167$	$0,167 < LCOM \leq 0,725$	$LCOM > 0,725$
NBD	$NBD \leq 1$	$1 < NBD \leq 3$	$NBD > 3$
RMD	$RMD \leq 0,467$	$0,467 < RMD \leq 0,750$	$RMD > 0,750$

Figura 4.1. Catálogo de valores referência para métricas de softwares orientados por objetos. Fonte: Filó et al. [2015]

4.5 Proposição de Estratégias para Detecção de *Bad Smells*

Foram propostas nesta dissertação, estratégias de detecção para cinco *bad smells* definidos por Fowler [1999]: *Large Class, Long Method, Data Class, Feature Envy* e *Refused Bequest*. O detalhamento dessas estratégias é realizado no Capítulo 5.

Para a definição da estratégia de detecção de cada *bad smell*: (i) foram analisados os aspectos estruturais do software que podem causá-lo, (ii) foram identificadas as métricas que avaliam tais fatores e (iii) foram definidas as estratégias de detecção aplicando-se os valores referência de tais métricas, de acordo com o catálogo de Filó et al. [2015] e (iv) foi verificado se as métricas selecionadas estavam disponíveis nos arquivos de métricas do *Qualitas.class* [Terra et al., 2013].

4.6 Definição de Medições para Análise dos Resultados

Os resultados obtidos nos experimentos foram analisados em termos de três métricas de avaliação amplamente utilizadas na literatura: *Recall*, Precisão e *F-measure* [Frakes & Baeza-Yates, 1992; Macia et al., 2012; Fontana et al., 2012; Paiva et al., 2015;

Fawcett, 2006; Shin & Williams, 2011; Sahin et al., 2014; Padilha et al., 2013]. Essas métricas foram escolhidas para a avaliação por serem medidas utilizadas para verificar a efetividade de resultados, além de serem complementares.

Recall pode ser considerada como uma medida de completude. Ela representa a habilidade para recuperação dos resultados relevantes de acordo com a consulta realizada, enquanto a Precisão verifica se os resultados estão dentro do esperado [Wikipédia, 2016; Fawcett, 2006].

No contexto desta dissertação, *Recall* é a capacidade das estratégias de detecção propostas encontrarem *bad smells*, e Precisão está relacionada à capacidade de detecção de verdadeiros positivos, que são as entidades consideradas *bad smells* tanto pelas estratégias de detecção propostas quanto pelas listas de referência utilizadas. Pode-se obter essas medidas a partir das fórmulas apresentadas a seguir.

$$Recall = \frac{VP}{VP + FN} \qquad Precisão = \frac{VP}{VP + FP}$$

Nas fórmulas de *Recall* e Precisão: Falsos Positivos (FP) incluem todas as entidades identificadas como *bad smells* nas estratégias de detecção propostas que não foram identificadas pelas ferramentas de detecção; Falsos Negativos (FN) referem-se a todas as entidades que não foram classificadas como *bad smells* nas estratégias de detecção, mas que, foram identificadas pelas ferramentas de detecção; Verdadeiros Positivos (VP) incluem todas as entidades identificadas como *bad smells* pelas estratégias de detecção que foram também identificadas pelas ferramentas.

A Figura 4.2 mostra uma visão em termos de conjunto para complementar o entendimento sobre os itens que compõem as fórmulas de *Recall* e Precisão. Na figura, o conjunto A corresponde aos valores obtidos pelas estratégias de detecção propostas e o conjunto B aos valores obtidos por listas de referência geradas por ferramentas de detecção de *bad smells* ou por especialistas.

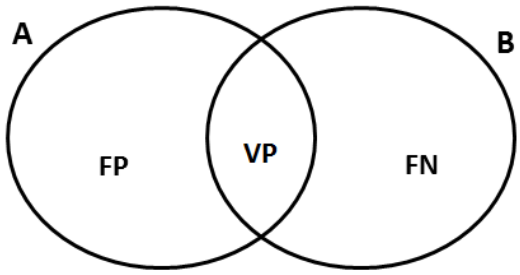

Figura 4.2. Visão de *Recall* e Precisão em termos de conjunto

A fórmula de *F-measure* também foi utilizada com a finalidade de verificar a exatidão dos resultados gerados. Ela leva em consideração os resultados obtidos com as fórmulas de *Recall* e Precisão, podendo ser interpretada como uma média ponderada entre essas duas medidas. A *F-measure* é dada pela média harmônica entre os valores de *Recall* e Precisão. A média harmônica é apropriada para situações onde é desejada a média das taxas ou percentagens [Wikipédia, 2016].

$$F - measure = 2 * (\frac{Precisão * Recall}{Precisão + Recall})$$

4.7 Considerações Finais

Este capítulo descreveu a metodologia de pesquisa usada para realização deste trabalho e suas principais etapas e atividades para que a qualidade de sistemas de softwares orientados por objeto seja avaliada de duas formas: (i) detecção de *bad smells* e (ii) predição de falhas. Na sequência, o Capítulo 5 apresenta as estratégias propostas neste trabalho de pesquisa para a detecção de *bad smells* e nos Capítulos 6, 7 e 8, são descritos os experimentos detalhando o trabalho realizado para responder as questões de pesquisa investigadas nesta dissertação.

Capítulo 5

Estratégias Propostas para Detecção de *Bad Smells*

Este capítulo apresenta as estratégias de detecção de *bad smells* propostas nesta dissertação. O termo "estratégia de detecção" pode ser definido como "uma expressão quantificável de uma regra", definida para "verificar se fragmentos do código-fonte estão em conformidade com essa regra definida" [Marinescu, 2004][Bertrán, 2009]. Os componentes utilizados para a definição de estratégias de detecção de *bad smells* são: mecanismos de filtragem de métricas e mecanismos de composição. O mecanismo de filtragem possibilita a redução do conjunto inicial de dados, de modo que se possa verificar uma característica especial de fragmentos que têm suas métricas capturadas. Os filtros permitem definir limites de valores para determinados aspectos. Por sua vez, os mecanismos de composição permitem combinar diferentes filtros, baseados em métricas de software distintas, por meio de conectivos lógicos, possibilitando, assim, a mescla de métricas para obter informações relevantes [Marinescu, 2004; Bertrán, 2009].

Para definir os mecanismos de filtragem e composição e propor as estratégias de detecção de *bad smells* utilizadas nesta dissertação, as métricas de software foram escolhidas de acordo com a sua disponibilidade no catálogo de valores referência definido por Filó et al. [2015]. Neste trabalho, foram definidas estratégias de detecção para cinco dos 22 *bad smells* propostos por Fowler [1999]. Os *bad smells* selecionados são aqueles possíveis de serem avaliados de acordo com as métricas disponíveis no referido catálogo de valores referência.

5.1 Métricas de Software Utilizadas nas Estratégias de Detecção

Nas estratégias de detecção propostas, foram utilizadas as métricas de software que possuem valores referências definidos por Filó et al. [2015] e que estão disponíveis no *Qualitas.class Corpus* [Terra et al., 2013].

As métricas a seguir foram aplicadas na definição das estratégias. A descrição dessas e de outras métricas consta no Capítulo 2 desta dissertação.

- DIT (*Depth in Inheritance Tree*): profundidade de uma classe na árvore de herança do software. Esta métrica é dada pela quantidade de classes que estão acima de determinada classe na hierarquia de heranças.

- LCOM (*Lack of Cohesion between Methods*): fornece uma medida da falta de coesão de uma classe.

- MLOC (*Method Lines of Code*): quantidade de linhas de código de um método.

- NBD (*Nested Blocks Depth*): quantidade máxima de blocos aninhados de um método.

- NSC (*Number of Children*): quantidade de classes filhas em relação a uma dada classe.

- NOF (*Number of Fields*): quantidade de atributos de uma classe.

- NOM (*Number of Methods*): quantidade de métodos em uma classe.

- VG (*McCabe's Complexity*): complexidade ciclomática de um método.

- WMC (*Weighted Methods per Class*): corresponde ao somatório da complexidade dos métodos da classe. Refere-se ao peso total de uma classe em relação aos pesos de cada método da classe.

- SIX (*Specialization Index*): avalia o quanto uma subclasse sobrescreve a superclasse.

5.2 Faixas e Valores Referência Utilizados nas Estratégias de Detecção

Para filtrar o universo de entidades presentes em um sistema de software e, assim, possibilitar identificação de instâncias de *bad smell*, cada métrica de software utilizada por uma estratégia de detecção deve estar relacionada a um valor limiar. Esse valor, denominado valor referência, define a faixa de valores de interesse em relação a uma métrica de software específica. Dessa forma, é possível utilizar tal métrica como apoio na identificação de algum *bad smell*. Por exemplo, para encontrar classes de um sistema que sejam muito grandes em termos de linhas de código, pode-se utilizar LOC como métrica de apoio em um filtro da estratégia de detecção. Porém, é preciso estabelecer um valor referência para LOC de forma que as classes que obedeçam a faixa de valores determinada pelo valor referência sejam identificadas corretamente. Um exemplo de valor referência poderia ser 1000, onde o filtro $LOC > 1000$ determinaria um limiar mínimo para que essa métrica indicasse classes grandes.

As estratégias de detecção para *bad smells* propostas nesta dissertação consideram as três faixas de valores referência definidas por Filó et al. [2015]: *Bom/Frequente*, *Regular/Ocasional* e *Ruim/Raro*. A faixa *Bom/Frequente* corresponde a valores com alta frequência, caracterizando os valores mais comuns da métrica, na prática. A faixa *Ruim/Raro* corresponde a valores com baixa frequência, e a faixa *Regular/ Ocasional* é intermediária, correspondendo a valores que não são nem muito frequentes nem raros. De acordo com Ferreira [2011], valores considerados frequentes em sistemas indicam que eles correspondem à prática comum no desenvolvimento de software de alta qualidade, o que serve como um parâmetro de comparação de um software com os demais. Por sua vez, valores pouco frequentes indicam situações não usuais na prática, portanto, pontos a serem considerados como críticos, que podem ser indicativos de problemas estruturais.

Nas estratégias propostas, considera-se que:

- *Bom*: limite superior da faixa *"Bom/ Frequente"*
- *Regular*: limite inferior da faixa *"Regular/ Ocasional"*
- *Ruim*: limite inferior da faixa *"Ruim/ Raro"*

5.3 Estratégias de Detecção de *Bad Smells* Propostas

Considerando as métricas e os valores referência apresentados por Filó et al. [2015], foi possível definir estratégias de detecção para os seguintes *bad smells* de Fowler [1999]: *Large Class, Long Method, Data Class, Feature Envy e Refused Bequest.* Para os *bad smells*: *Duplicated Code, Message Chains, Parallel Inheritance Hierarchies, Speculative Generality, Swith Statements, Temporary Field, Primitive Obsession, Inappropriate Intimacy e Comments* não foi possível a definição de estratégias visto que as métricas utilizadas por Filó et al. [2015] não apoiam sua detecção.

No caso dos *bad smells Alternative Classes with Different Interfaces e Incomplete Library Class*, embora as métricas utilizadas por Filó et al. [2015] permitam a definição de estratégias de detecção, eles não foram incluídos neste estudo. O motivo principal é que não foram encontradas ferramentas que avaliem tais *bad smells* [Fernandes et al., 2016].

Os demais *bad smells* definidos por Fowler [1999] não foram incluídos por não ter sido possível a utilização de ferramentas que os detectassem, o que inviabilizaria parte da avaliação das estratégias. São eles:

- *Lazy Class* – quatro ferramentas foram encontradas na literatura mas não estão disponíveis para *download* [Fernandes et al., 2016]. E, uma ferramenta, chamada *True Refactor* foi citada por Fernandes et al. [2016] como a única ferramenta disponível para *download* para esse *bad smell* mas não foi encontrada.

- *Middle Man* – De acordo com a literatura apenas uma ferramenta está disponível para a detecção desse *bad smell*, *Code Bad Smell Detector* [Fernandes et al., 2016]. Foi possível fazer o *download* mas não foi possível executá-la devido a um problema no arquivo de configuração.

- *Data Clumps* – Das cinco ferramentas disponíveis para esse *bad smell*, três necessitam de licença para utilização, *InCode, InFusion, IntelliJ IDEA*. *Code Bad Smell Detector* apresentou problema no arquivo de configuração, e *Stench Blossom* apresentou problemas de incompatibilidade com a versão do *Eclipse* utilizada nesse estudo.

- *Long Paramenter List* – De acordo com Fernandes et al. [2016] esse *bad smell* possui na literatura nove ferramentas de detecção. Porém, cinco não estão disponíveis para *download*, *IntelliJ IDEA* precisa de licença; *Gendarme* não realiza

detecção em sistemas Java; *Checkstyle* não retornou nesse estudo instâncias desse *bad smell* para os sistemas selecionados; e por fim, a ferramenta *PMD* apresentou falhas ao tentar exportar os resultados de alguns sistemas.

- *Shotgun Surgery* – Para esse *bad smell* três ferramentas não estão disponíveis para *download* [Fernandes et al., 2016]. E dentre as disponíveis, a *IntelliJ IDEA* precisa de licença e a *iPlasma* apresentou problemas de exportação dos resultados.

Para cada estratégia de detecção proposta são apresentadas a seguir: uma descrição resumida do *bad smell* de Fowler [1999] a qual ela se refere, as métricas utilizadas e a estratégia em si.

Data Class: refere-se a classes que contém somente atributos e métodos *get() e set()*. São classes que funcionam como um simples agregador de dados, geralmente sem processamento complexo.

As métricas utilizadas para compor a estratégia de detecção desse *bad smell* são:

- NSC: considerando que uma *Data Class* emula um tipo primitivo de dados, assume-se que esse tipo de classe tende a não ter subclasses. Espera-se que uma *Data Class* ofereça somente acesso aos seus atributos, por meio de *getters e setters*. Portanto, considera-se que valores baixos de NSC, em conjunto com outras características, possam auxiliar a identificar *Data class*.
- DIT: Uma classe de dados é auto-contida, provê somente acesso aos seus atributos. Não precisa herdar responsabilidade de uma superclasse. Partindo das mesmas considerações descritas para NSC, considera-se também que uma *Data Class*, em geral, não herda responsabilidades de superclasses. Dessa forma, considera-se que um DIT baixo, associado a outras características, possa auxiliar a identificar *Data Class*.
- NOF: como a classe somente armazena dados, assume-se que uma quantidade *Regular* ou *Ruim* de atributos (NOF) indica a ocorrência de *Data Class*.

A estratégia de detecção para *Data Class* é definida conforme a Figura 5.1: NSC e DIT devem ter valor dentro do respectivo intervalo *Bom* da métrica, incluindo o limite superior desse intervalo; NOF deve ter valor dentro dos intervalos *Regular* ou *Ruim*, incluindo o limite inferior do intervalo *Regular* da métrica.

Feature Envy: refere-se a um trecho de código pertencente a uma classe que está mais interessado nos dados/processamentos de outra classe.

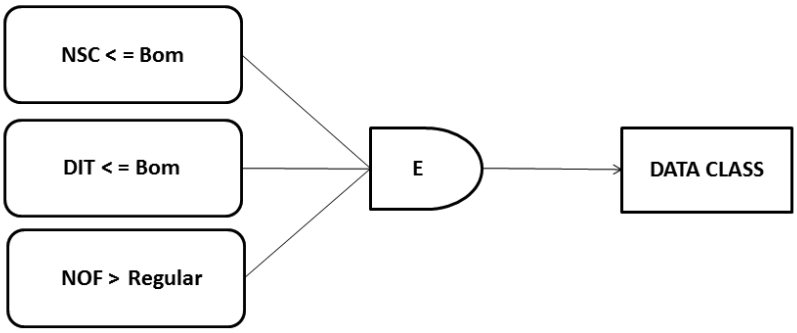

Figura 5.1. Estratégia proposta para detecção de *Data Class*

As métricas utilizadas para compor a estratégia de detecção desse *bad smell* são:

- LCOM: é a única métrica do catálogo de Filó [2014] capaz de detectar, de alguma forma, o interesse de uma classe por dados de outra classe. Mesmo que indiretamente, a baixa coesão de uma classe pode indicar que ela está interessada em dados e/ou processamentos de outras classes, pelo fato de ela implementar vários interesses divergentes.

  A Figura 5.2 mostra a estratégia de detecção de *Feature Envy*.

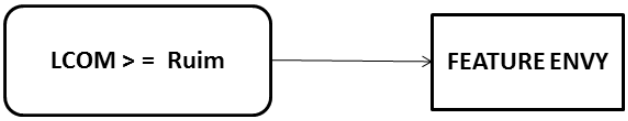

Figura 5.2. Estratégia proposta para detecção de *Feature Envy*

***Large Class*:** refere-se a classes que possuem muitas responsabilidades, processamentos e atributos. Elas tendem a centralizar a inteligência do sistema, realizando uma quantidade de trabalho excessivo e se tornando uma agregação de diferentes abstrações. Isso está em desacordo com um dos princípios básicos da orientação por objetos, que diz que uma classe deve ter uma única responsabilidade. Algumas características desse tipo de classe é que elas são grandes, complexas e possuem baixa coesão.

As métricas utilizadas para compor a estratégia de detecção desse *bad smell* são:

- NOF, NOM: uma quantidade alta de atributos (NOF) e de métodos (NOM) aponta excessivo conhecimento (atributos) e processamento (métodos) da classe.
- WMC: considera-se que quanto maior o peso (*weight*) da classe, maior a sobrecarga de operações da mesma. WMC pode indicar muitas responsabilidades da classe.
- LCOM: falta de coesão entre os métodos da classe indica sobrecarga de responsabilidades. Portanto, LCOM alto sugere que a classe seja uma instância de *Large Class*.

A Figura 5.3 mostra a estratégia de detecção proposta para *Large Class*.

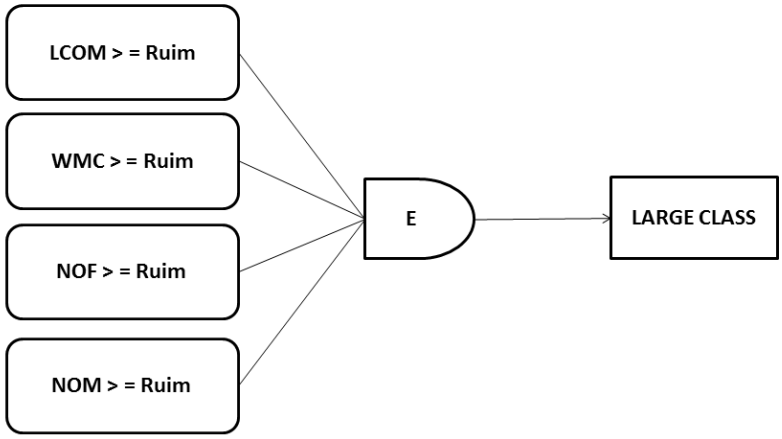

Figura 5.3. Estratégia proposta para detecção de *Large Class*

***Long Method*:** refere-se a métodos complexos, extensos e/ou com várias responsabilidades. São métodos nos quais, no processo natural da evolução de software, são adicionadas mais e mais funcionalidades, tornando-o de difícil compreensão e manutenção.

As métricas utilizadas para compor a estratégia de detecção deste *bad smell* são:

- MLOC: métodos longos geralmente possuem uma quantidade de linhas de código alta.
- VG: a complexidade ciclomática do método alta indica alta complexidade do método.
- NBD: um valor de aninhamento de blocos alto em um método é um indício de processamento complexo/excessivo.

A estratégia de detecção proposta para *Long Method* é mostrada na Figura 5.4.

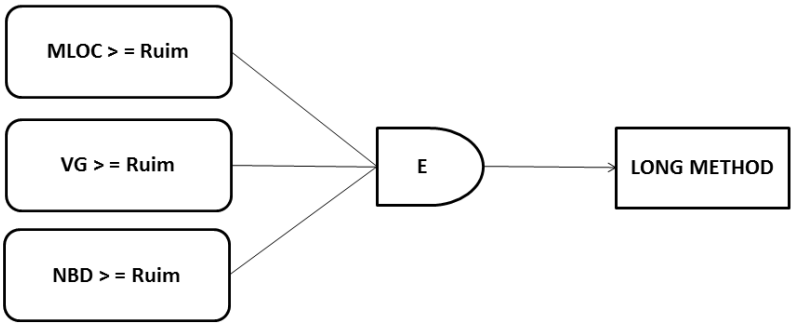

Figura 5.4. Estratégia proposta para detecção de *Long Method*

***Refused Bequest*:** refere-se a subclasses que possuem herança de métodos e dados mas que utilizam poucos deles. Ou seja, a classe filha "despreza" os atributos e métodos da superclasse.

As métricas utilizadas para compor a estratégia de detecção deste *bad smell* são:

- SIX: quanto maior o grau de sobrescrita que a classe "filha" possui em relação à superclasse, maiores as chances da classe "filha" estar recusando as responsabilidades providas por meio da herança. Dessa forma, utilizou-se SIX como medida de uma característica particular e mensurável de *Refused Bequest*.

A Figura 5.5 mostra a estratégia de detecção proposta para *Refused Bequest*.

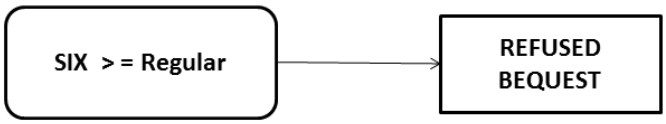

Figura 5.5. Estratégia proposta para detecção de *Refused Bequest*

5.4 Considerações Finais

Este capítulo apresentou as estratégias de detecção definidas para cinco *bad smells* propostos por Fowler [1999], com base nos valores referência do catálogo de Filó et al.

[2015]. Filó et al. [2015] definiu estratégias para apenas dois *bad smells: God Class e Long Method*. A avaliação das estratégias propostas neste trabalho, permite avaliar a aplicabilidade dos valores referência de métricas de software na detecção dos seguintes *bad smells: Large Class, Long Method, Data Class, Feature Envy e Refused Bequest*. A aplicação dos valores referência de Filó et al. [2015] também permitem a definição de estratégias para outros *bad smells: Lazy Class, Middle Man, Data Clumps, Long Paramenter List, Shotgun Surgery, Alternative Classes with Different Interfaces, Incomplete Library Class*. Porém, tais estratégias não foram incluídas neste trabalho uma vez que a avaliação delas foi limitada devido: (i) ausência de ferramentas disponíveis para a detecção desses *bad smells*, (ii) necessidade de licença, (iii) erro no arquivo de configuração ou incompatibilidade com a versão da *IDE Eclipse* utilizada no estudo, (iv) problema na exportação dos resultados. A avaliação das estratégias de detecção propostas é apresentada nos Capítulos 6 e 7.

Capítulo 6

Valores Referência e Detecção Automatizada de *Bad Smells*

O objetivo deste capítulo é avaliar a capacidade de os valores referência propostos por Filó et al. [2015] auxiliarem na identificação dos *bad smells* definidos por Fowler [1999]. A identificação de classes e métodos com problemas estruturais reflete o panorama de qualidade deteriorada do software. Dessa forma, este estudo visa avaliar se os valores referência de métrica de softwares orientados por objetos podem auxiliar na garantia da qualidade dos softwares.

Esse capítulo investiga as questões de pesquisa *QP1* deste trabalho, especificamente a *QP1.1*, que são:

QP1. *Os valores referência de métricas de software orientados por objetos auxiliam a identificar bad smells?*

QP1.1. *Qual é a eficácia da detecção de bad smells utilizando-se estratégias baseadas nos valores referência e tomando-se como base os resultados gerados por ferramentas de detecção de bad smells*?

A fim de apoiar a definição de etapas e execução do estudo realizado neste trabalho para responder à questão de pesquisa *QP1.1*, apresenta-se, a seguir, a definição do escopo do estudo baseado em Wohlin et al. [2012].

Analisar os valores referência propostos por Filó et al. [2015]

A fim de verificar sua utilidade em detectar *bad smells* por meio de estratégias de detecção

Em relação a *Recall*, Precisão e *F-measure* das estratégias de detecção para diferentes *bad smells: Large Class, Long Method, Feature Envy, Data Class e Refused Bequest*

Sob o ponto de vista de listas de referências geradas por ferramentas automatizadas e especialistas

No contexto de um conjunto de sistemas de código aberto desenvolvidos em Java selecionados do *Qualitas.class Corpus* 2013.

Para a realização deste estudo, foram seguidas as etapas apresentadas na Figura 6.1. O detalhamento dessas etapas foi apresentado no Capítulo 4.

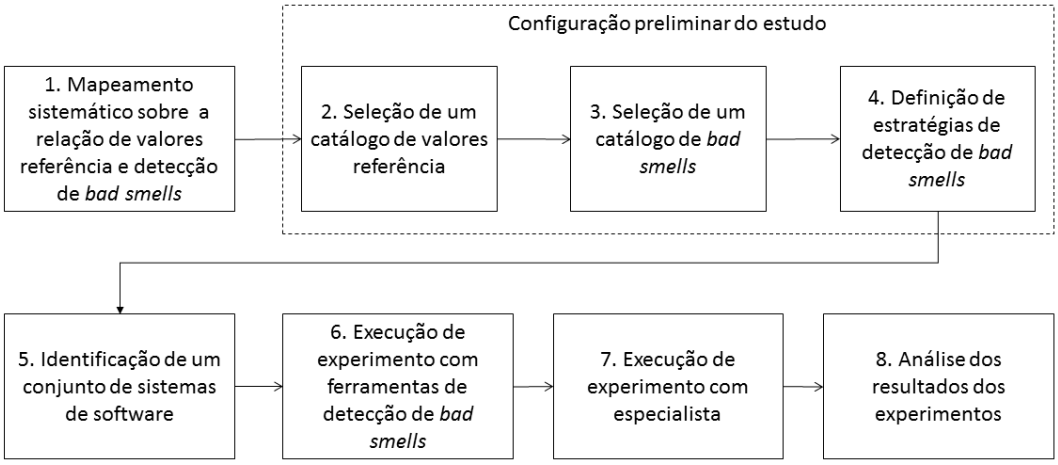

Figura 6.1. Etapas do estudo sobre aplicação dos Valores Referência x *Bad Smells*

Nesse estudo os resultados obtidos por meio das estratégias propostas foram comparados com os resultados providos por duas ferramentas de detecção de *bad smells* utilizadas na literatura: *JSpIRIT e JDeodorant* [Paiva et al., 2015; Filó, 2014; Filó, 2014; Fernandes et al., 2016].

A ferramenta *JSpIRIT* [Vidal et al., 2014] é um *plugin* para a *Eclipse IDE* que identifica *bad smells* em software *Java* por meio de uma abordagem baseada em métri-

cas. A ferramenta foi utilizada nesse experimento para identificar a presença dos *bad smells: Large Class, Long Method, Data Class, Feature Envy e Refused Bequest.*

A ferramenta *JDeodorant* [Tsantalis et al., 2008] é um *plugin* de código-fonte aberto para *Eclipse* que identifica quatro *bad smells* em software *Java*: *God Class (similar a Large Class), God Method (Similar a Long Method), Feature Envy e Switch Statement.* As técnicas de detecção aplicadas pela ferramenta baseiam-se em oportunidades de refatoração para *God Class e Feature Envy* e técnicas *slicing*, para detectar *God Method* [Fontana et al., 2015]. Nesse experimento, a *JDeodorant* foi utilizada para detectar *Large Class, Long Method e Feature Envy.*

Para implementar os *scripts* com as estratégias de detecção propostas no Capítulo 5 dessa dissertação foi usada uma ferramenta denominada *FindSmells* [Sousa et al., 2016]. *FindSmells* é uma ferramenta de detecção de *bad smells* que permite ao usuário selecionar um ou mais arquivo em formato XML contendo as medições de um software. Ela processa esses arquivos e insere as medições em um banco de dados. Após o usuário selecionar o sistema e o *bad smell* que deseja identificar, *FindSmells* executa a estratégia de detecção que foi implementada para aquele *bad smell.* Nesse processo, ela faz a filtragem de métodos, classes e pacotes que possuem medições anômalas de métricas de softwares orientados por objetos no contexto do processo de medição de software. Medições anômalas são aquelas que se afastam significativamente do que é comum, podendo indicar problemas de qualidade no artefato medido [Sommervile, 2011]. *FindSmells* também permite a edição das estratégias de detecção já implementadas bem como possibilita a implementação de novas estratégias de detecção.

As ferramentas de detecção bem como as estratégias propostas foram executadas nos 12 sistemas selecionados do *Qualitas.class Corpus 2013* descrito no Capítulo 4 dessa dissertação. A Tabela 6.1 apresenta a lista dos 12 sistemas de pequeno a médio porte selecionados aleatoriamente para realização desse estudo. Os resultados da *JSpIRIT e JDeodorant* não são tomados como corretos neste trabalho. O motivo para isso é que os resultados reportados por essas ferramentas são diferentes entre si. Além disso, não há estudos prévios indicando a acurácia de tais ferramentas. Por outro lado, são ferramentas utilizadas em estudos prévios a respeito de *bad smells* [Filó, 2014; Filó et al., 2015; Fernandes et al., 2016]. Dessa forma, a análise realizada aqui fornece uma visão de como os resultados das estratégias propostas se comparam aos dessas duas ferramentas.

Os resultados obtidos neste estudo são apresentados e analisados nas próximas seções: Seção 6.1 apresenta a análise com a ferramenta *JSpIRIT.* Seção 6.2 apresenta a análise com a ferramenta *Jdeodorant.* Seção 6.3 apresenta as ameaças à validade dos experimentos. Seção 6.4 destaca as lições aprendidas desse estudo.

Tabela 6.1. Sistemas selecionados do *Qualitas.class Corpus* 2013 para estudo de VR x *Bad Smells*

#	Sistema	Tamanho	Nº de Classes	Nº de Métodos
1	aoi-2.8.1	9.4 MB	841	6915
2	checkstyle-5.6	53 MB	560	2896
3	cobertura-1.9.4.1	11 MB	174	3520
4	displaytag-1.2	10 MB	336	1728
5	itext-5.0.3	7.8 MB	587	5982
6	jedit-4.3.2	15 MB	1183	7315
7	joggplayer-1.1.4s	7.2M	363	2299
8	jsXe-04_beta	17 MB	270	1445
9	pmd-4.2.5	12 MB	914	5958
10	quartz-1.8.3	12 MB	316	2923
11	squirrel_sql-3.1.2	13 MB	169	689
12	webmail-0.7.10	12 MB	129	1091

6.1 Análise com a Ferramenta *JSpIRIT*

Os resultados obtidos pelas estratégias propostas nesta dissertação foram comparados com os resultados obtidos pela *JSPiRIT* em termos de *Recall*, Precisão e *F-measure*. No caso do *bad smell Data Class* os resultados foram computados com base em somente 8 dos 12 sistemas pois, para os demais sistemas, a *JSpIRIT* não encontrou nenhuma instância de *Data Class*, ou houve limite de memória excedido pela ferramenta.

6.1.1 Análise de *Recall*

No contexto desse experimento, *Recall* mede a capacidade das estratégias de detecção propostas encontrarem instâncias de *bad smells* considerando o "todo" isto é, o sistema de software sob análise. Assim, quanto maior o valor de *Recall* para uma estratégia, maior é a sua capacidade de encontrar instâncias do *bad smell* que se propõe a encontrar.

A Figura 6.2 apresenta a distribuição de *Recall* por *bad smell*. A figura mostra os resultados de distribuição para *Large Class (LC), Long Method (LM), Data Class (DC), Feature Envy (FE) e Refused Bequest (RB)*, respectivamente. Para apoiar a compreensão dessa figura, a Tabela 6.2 provê a análise descritiva em termos dos percentis de cada *boxplot*. Uma discussão acerca dos resultados exibidos na Figura 6.2 é apresentada a seguir.

Análise de *Recall* Por *Bad Smell*. Analisando o gráfico de distribuição de *Recall* por *bad smell*, Figura 6.2, nota-se que as medianas estão acima de 40% de *Recall* para 4

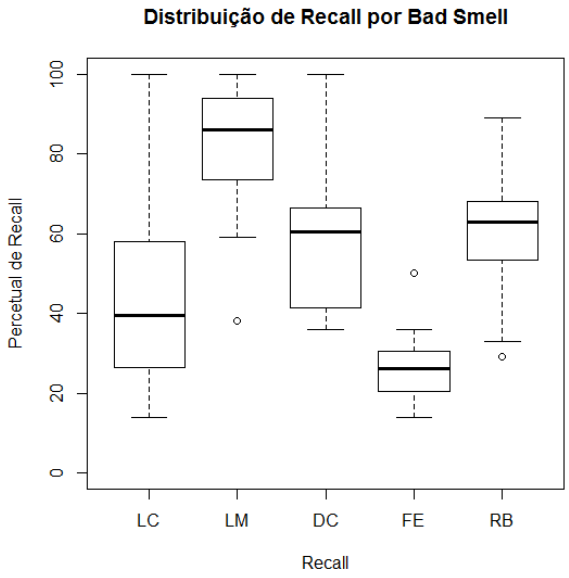

Figura 6.2. Distribuição de *Recall* por *Bad Smell* para *JSpIRIT*

Tabela 6.2. Tabela de análise descritiva de *Recall* por *Bad Smell* para *JSpIRIT*

Bad Smell	**0%**	**25%**	**50%**	**75%**	**100%**
Large Class	14,0	27,7	39,5	52,5	100,0
Long Method	38,0	76,7	86,0	92,5	100,0
Data Class	36,0	42,2	60,5	65,2	100,0
Feature Envy	14,0	20,7	26,0	29,7	50,0
Refused Bequest	29,0	53,5	63,0	68,0	89,0

dos 5 *bad smells* analisados: *Large Class, Long Method, Data Class e Refused Bequest*. Isso aconteceu porque para 6 dos 12 sistemas analisados nesse experimento, ou seja, para metade do total de sistemas, o *Recall* foi maior ou igual a 40% para cada *bad smell*. Este resultado é razoável se for considerado o contexto do estudo: detecção de *bad smells* em código-fonte, pois essa não é uma tarefa trivial. A literatura reporta que, mesmo em sistemas de pequeno-porte, várias instâncias de *bad smells* podem ser encontradas [Macia et al., 2012]. Assim, um *Recall* de ao menos 40% significa que, em um universo extenso de possíveis instâncias, as estratégias de detecção foram capazes de encontrar uma quantidade significativa delas.

Observa-se também que a distribuição de *Recall* para *Feature Envy* tem mediana de 26%, e que o terceiro quartil (75%) não está muito distante disso, com valor próximo de 30%. Uma justificativa para isso é a carência de métricas no catálogo de Filó et al. [2015] adequadas à detecção desse *bad smell*. Por exemplo, utilizou-se somente LCOM na estratégia proposta, por falta de métricas que meçam acoplamento

entre classes. Outro dado importante é que foi obtido *Recall* de 100%, para pelo menos um sistema, considerando os seguintes *bad smells: Large Class, Long Method e Data Class*. Além disso, o *bad smell Refused Bequest*, obteve *Recall* de 90% para pelo menos um dos sistemas analisados.

Análise de *Recall* Considerando Todos os *Bad Smells*. A Figura 6.3 apresenta a distribuição de *Recall* considerando todos os *bad smells* analisados neste experimento. Esta análise tem como objetivo comparar o conjunto de estratégias definido com os resultados reportados pelas ferramentas independente do *bad smell* identificado. Para apoiar a compreensão dessa figura, a Tabela 6.3 provê a análise descritiva em termos dos percentis do *boxplot*. Uma discussão acerca da Figura 6.3 é apresentada a seguir.

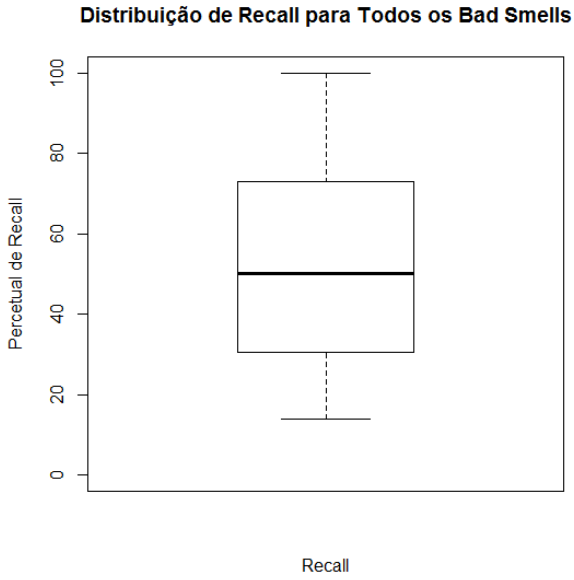

Figura 6.3. Distribuição de *Recall* para todos os *Bad Smells* para *JSpIRIT*

Tabela 6.3. Análise descritiva de *Recall* considerando todos os *Bad Smells* para *JSpIRIT*

Percentil	**0%**	**25%**	**50%**	**75%**	**100%**
Recall	14,0	30,5	50,0	73,0	100,0

Observando a Figura 6.3, tem-se uma mediana de 50% de *Recall*, o que é razoável considerando que mesmo sistemas pequenos podem conter uma quantidade elevada de instâncias de *bad smells*. Além disso, não foi obtido *Recall* menor do que 14% para

nenhum dos *bad smells* avaliados. Para 25% dos sistemas avaliados, foi obtido um *Recall* significativamente alto, com valor maior ou igual a 73%. Note-se também que, *Feature Envy* foi o único *bad smell* com *Recall* menor do que 40%. Provavelmente os resultados obtidos para este *bad smell* contribuíram para a diminuição dos valores na distribuição de *Recall* em relação a todas as estratégias propostas. Portanto, desconsiderando o tipo de *bad smell* que as estratégias se propõem a identificar, tais estratégias mostraram-se suficientemente eficazes em termos de *Recall*, ainda que os resultados de *Feature Envy* tenham sido baixos em relação aos demais *bad smells*.

6.1.2 Análise de Precisão

Diferentemente do *Recall*, a Precisão mede a capacidade das estratégias de detecção propostas encontrarem instâncias verdadeiras de *bad smells*. Em outras palavras, enquanto o *Recall* mede a quantidade de resultados obtidos, a Precisão mede a qualidade dos resultados obtidos. Instâncias verdadeiras neste estudo referem-se àquelas identificadas pela ferramenta *JSpIRIT*. Todavia, conforme descrito anteriormente, não se tem como premissa, neste trabalho, que os dados reportados pelas ferramentas são de fato verdadeiros. Os resultados das ferramentas são utilizados neste trabalho para fins de comparação com as estratégias propostas.

Análise de Precisão por *Bad Smell*. A Figura 6.4 apresenta a distribuição de Precisão por *Bad Smell*. Nessa figura são apresentados os resultados de distribuição para *Large Class (LC), Long Method (LM), Data Class (DC), Feature Envy (FE) e Refused Bequest (RB)*. A Tabela 6.4 provê a análise descritiva em termos dos percentis de cada *boxplot*. Uma discussão acerca dos resultados ilustrados na Figura 6.4 é apresentada a seguir.

Tabela 6.4. Tabela de análise descritiva de Precisão por *Bad Smell* para *JSpIRIT*

Bad Smell	**0%**	**25%**	**50%**	**75%**	**100%**
Large Class	50,0	63,0	78,5	89,5	100,0
Long Method	9,0	26,0	34,5	46,0	56,0
Data Class	7,0	7,7	9,5	12,0	28,0
Feature Envy	37,0	43,2	46,5	56,0	62,0
Refused Bequest	5,0	8,0	13,0	17,5	41,0

Observando-se a Figura 6.4, distribuição de Precisão por *bad smell*, é possível perceber que as medianas de percentual de Precisão estão acima de 40% para dois *bad smells: Large Class e Feature Envy*. Verificando-se os dados dos sistemas para tais *bad*

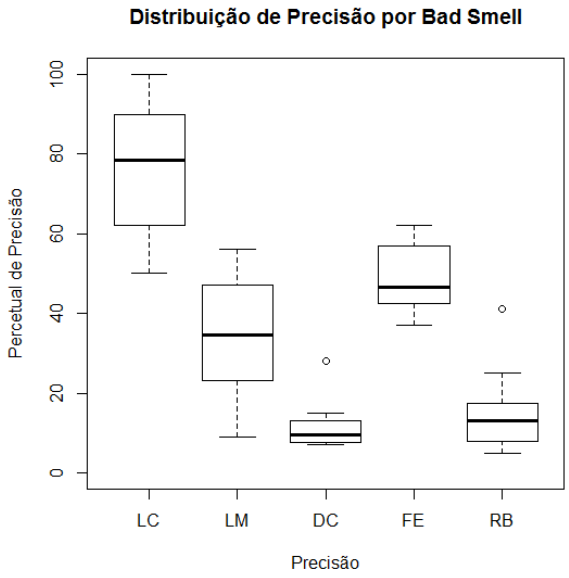

Figura 6.4. Distribuição de Precisão por *Bad Smell* para *JSpIRIT*

smells, nota-se que 6 dos 12 sistemas analisados possuem Precisão maior ou igual a 40%. Além disso, para *Long Method*, a Precisão é pouco maior que 30%.

Análise de Precisão Considerando Todos os *Bad Smells*. A Figura 6.5 apresenta a distribuição de Precisão considerando todos os cinco *bad smells* analisados. A Tabela 6.5 provê a análise descritiva em termos dos percentis do *boxplot*. Uma discussão acerca dos resultados mostrado na Figura 6.5 é apresentada a seguir.

Tabela 6.5. Análise descritiva de Precisão considerando todos os *Bad Smells* para *JSpIRIT*

Percentil	**0%**	**25%**	**50%**	**75%**	**100%**
Precisão	5,0	13,5	41,0	56,0	100,0

Observa-se Precisão de 41% para metade dos sistemas analisados, independente do *bad smell* detectado. Esse resultado pode ser consequência das baixas precisões obtidas para *Long Method, Data Class e Refused Bequest*, com medianas próximas de 30%, 10% e 15%, respectivamente. Os valores de Precisão observados foram inversamente proporcionais aos valores de *Recall*. Em outras palavras, quanto maior a quantidade de instâncias retornadas pelas estratégias propostas (*Recall*), menor foi o percentual de acerto das estratégias em relação às listas de referência, ou seja, a Precisão.

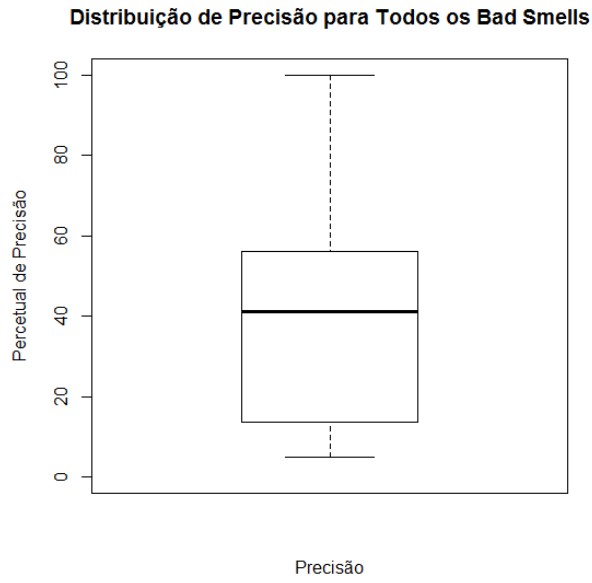

Figura 6.5. Distribuição de Precisão para todos os *Bad Smells* para *JSpIRIT*

6.1.3 Análise de *F-measure*

A *F-measure* foi utilizada com a finalidade de verificar a exatidão dos resultados gerados. Ela leva em consideração os resultados obtidos com as fórmulas de *Recall* e Precisão, podendo ser interpretada como uma média ponderada entre essas duas medidas.

Análise de *F-measure* por *Bad Smell*. A Figura 6.6 apresenta a distribuição de *F-measure* obtida por *bad smell*. Ela apresenta os resultados de distribuição para *Large Class (LC), Long Method (LM), Data Class (DC), Feature Envy (FE) e Refused Bequest (RB)*. A Tabela 6.6 provê a análise descritiva em termos dos percentis de cada *boxplot*.

Tabela 6.6. Tabela de análise descritiva de *F-measure* por *Bad Smell* para *JSpIRIT*

Bad Smell	**0%**	**25%**	**50%**	**75%**	**100%**
Large Class	22,0	42,3	51,5	63,0	78,0
Long Method	15,0	39,5	47,0	59,3	72,0
Data Class	12,0	13,5	16,0	20,5	44,0
Feature Envy	20,0	29,5	34,0	38,3	52,0
Refused Bequest	9,0	13,5	22,0	26,5	56,0

Pode-se observar na Figura 6.6, que somente para dois *bad smells, Large Class e*

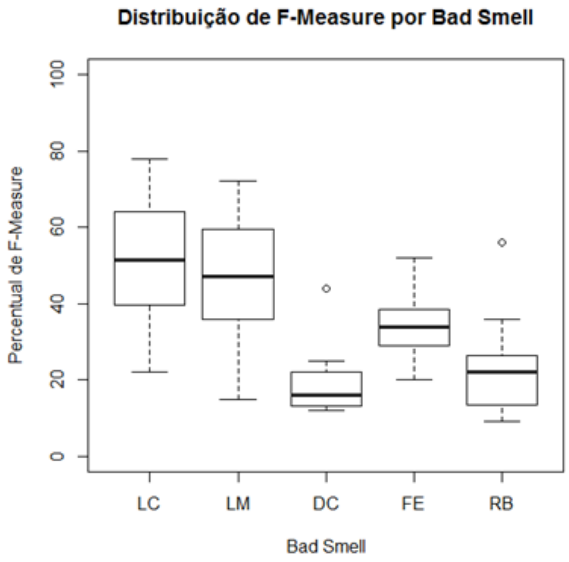

Figura 6.6. Distribuição de *F-measure* por *Bad Smell* para *JSpIRIT*

Long Method, a mediana obtida para *F-measure* está próxima de 50%. Isso significa que, balanceando-se as medidas de *Recall* e Precisão, tem-se uma acurácia significativa e portanto, as estratégias de detecção propostas para esses *bad smells* se mostraram compatíveis com os resultados da *JSpIRIT*. Nota-se que o valor obtido para mediana no caso de *Feature Envy* não foi tão abaixo dos valores para *Large Class e Long Method*. *Data Class e Refused Bequest* tiveram resultados menores ainda tendo como base o conjunto de sistemas selecionados.

Conforme mostra a Figura 6.6, foram alcançados coeficientes de até 78% e 72% para *Large Class e Long Method*, respectivamente. Ou seja, ao balancear *Recall* e Precisão, os resultados mantiveram-se expressivos para esses dois *bad smells*. Para os demais *bad smells*, observa-se que *F-measure* foi baixo. Nesses casos, o *Recall* foi, em geral, baixo, mas a Precisão foi significativa. Isso indica que os resultados gerados pelas estratégias propostas para esses *bad smells* tendem a divergir em relação aos resultados da *JSpIRIT*. Ressalta-se que a análise *F-measure* pode ter sido impactada pelo fato de que somente 8 dos 12 sistemas tiveram instâncias de *Data Class* encontradas pela *JSpIRIT*.

Análise de *F-measure* Considerando Todos os *Bad Smells*. A Figura 6.7 apresenta a distribuição de *F-measure* em relação a todos os *bad smells* analisados: *Large Class (LC), Long Method (LM), Data Class (DC), Feature Envy (FE) e Refused Be-*

quest (RB). A Tabela 6.7 provê a análise descritiva em termos dos percentis do *boxplot*.

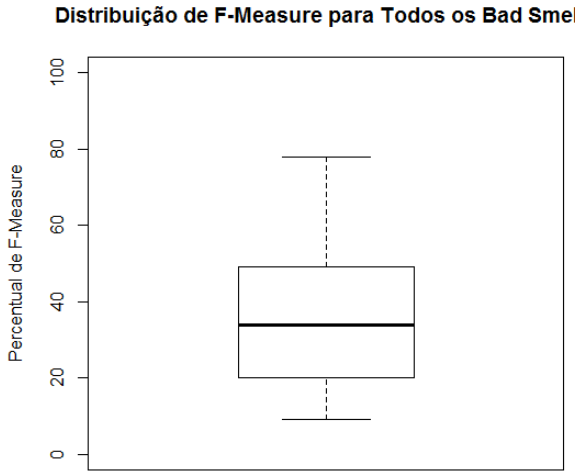

Figura 6.7. Distribuição de *F-measure* para todos os *Bad Smells* para *JSpIRIT*

Tabela 6.7. Análise descritiva de *F-measure* considerando todos os *Bad Smells* para *JSpIRIT*

Percentil	**0%**	**25%**	**50%**	**75%**	**100%**
F-measure	9,0	20,0	34,0	49,0	78,0

De acordo com a Tabela 6.7 tem-se que, para metade dos sistemas avaliados, obteve-se *F-measure* menor que 34%. De fato, a distribuição de *F-measure* mostrada na Figura 6.7 concentra-se em valores menores em comparação às distribuições das Figuras 6.3 e 6.5 que mostram os dados de *Recall* e de Precisão, respectivamente. Essa diferença pode ser justificada por dois fatores principais: (i) para 3 dos 5 *bad smells* avaliados, obteve-se Precisão muito baixa; (ii) além disso, obteve-se *Recall* baixo para 2 dos 5 *bad smells* avaliados. A *F-measure* pondera *Recall* e Precisão em uma medida de acurácia única. Portanto, é esperada uma concentração baixa dos valores de *F-measure* independente do *bad smell* que as estratégias se propõem a identificar. Esse resultado indica que, no geral, os resultados das estratégias de detecção propostas, quando analisadas como um todo, têm baixa coincidência com os resultados de *JSpIRIT*. Porém, ressalta-se que esse resultado não reflete as situações de *Recall* e Precisão analisadas isoladamente. Por exemplo, para *Large Class e Long Method*, obteve-se distribuições razoáveis de *Recall* e Precisão, considerando o contexto de detecção de *bad smells*.

6.1.4 Inspeção Manual dos Resultados da Ferramenta *JSpIRIT*

Trabalhos anteriores reportam ineficiências em ferramentas para detecção automatizada de *bad smells* em código-fonte [Bellon et al., 2007; Fernandes et al., 2016; Liu et al., 2012]. Isso pode ser causado, entre outros fatores, pela forma como os valores referência e métricas utilizadas pelas ferramentas na composição de estratégias de detecção são definidos. Essas definições podem variar de acordo com o contexto dos sistemas analisados, e isso pode impactar na qualidade dos resultados.

A fim de minimizar uma das principais ameaças desse experimento que foi a utilização dos resultados da *JSpIRIT* como lista de referência de *bad smells*, foi conduzida uma validação dos resultados com apoio de um especialista que possui conhecimentos em programação orientada por objetos e que conhece as definições dos *bad smells* de Fowler [1999].

O especialista realizou a validação dos resultados providos pela *JSpIRIT* em 2 dos 12 sistemas de software avaliados nesse experimento: *Squirrel SQL e Webmail*. Esses sistemas foram escolhidos por serem os menores dentre o conjunto analisado, para assim, viabilizar a análise manual das instâncias de *bad smells* reportadas pela ferramenta, com base no código-fonte dos sistema, sob o ponto vista do especialista.

A Tabela 6.8 apresenta os resultados obtidos por meio da análise com validação do especialista. Para fins de comparação, são apresentados os valores de *Recall* e Precisão referentes aos dados não-validados (NV) e aos dados após validação realizada pelo especialista (V). A tabela também mostra a diferença (DIF) entre os percentuais de *Recall* e Precisão validados e não-validados. O *bad smell Data Class* foi descartado dessa análise, porque a *JSpIRIT* não retornou instâncias desse *bad smell* para os dois sistemas selecionados para avaliação. Além disso, a ferramenta não retornou instâncias de *Refused Bequest* para o sistema *Squirrel SQL*. Porém, considerando que a ferramenta reportou resultados para o outro sistema avaliado, *Webmail*, esse *bad smell* não foi descartado da análise.

Tabela 6.8. Comparação dos resultados do Especialista com as Estratégias de Detecção Propostas

Sistema		***Large Class***			***Long Method***			***Feature Envy***			***Refused Bequest***		
		NV	**V**	**DIF**	**NV**	**V**	**DIF**	**NV**	**V**	**DIF**	**NV**	**V**	**DIF**
Squirrel	***Recall***	100%	100%	0%	100%	100%	0%	50%	60%	10%	-	-	-
	Precisão	60%	40%	-20%	17%	17%	0%	55%	30%	-25%	-	-	-
Webmail	***Recall***	15%	50%	35%	100%	100%	0%	27%	33%	6%	57%	80%	23%
	Precisão	100%	50%	-50%	56%	41%	-15%	47%	24%	-23%	14%	14%	0%

Nota-se que a DIF apresentou valores positivos para *Recall* no caso de três *bad*

smells: Large Class, Feature Envy e Refused Bequest. Isso significa que a validação do especialista identificou falsos positivos nos resultados gerados pela ferramenta. A remoção dessas instâncias identificadas pelo especialista gerou aumento do *Recall*. Além disso, um DIF negativo também foi obtido para Precisão no caso de três *bad smells: Large Class, Long Method e Feature Envy*. Nesse caso, conclui-se que, em algumas situações, o especialista não identificou instâncias verdadeiras nos resultados gerados pela *JSpIRIT*. Esse fato impactou negativamente no resultado da Precisão. Considerando a Tabela 6.8 e os *bad smells Large Class e Feature Envy*, observa-se também um aumento nos percentuais de *Recall* acompanhado por uma diminuição de Precisão.

A Figura 6.8 apresenta a distribuição de *Recall* considerando todos os *bad smells* cujas instâncias foram validadas pelo especialista em relação aos dois sistemas avaliados. A Tabela 6.9 provê a análise descritiva em termos dos percentis do *boxplot*. Observa-se que foi obtida uma mediana de 80% para *Recall*. Além disso, nota-se que $\frac{3}{4}$ dos resultados de *Recall* são maiores ou iguais a 55%. Tais resultados são bastante expressivos considerando as dificuldades inerentes à detecção de *bad smells* em sistemas de software. Isso pois, mesmo em sistemas pequenos, pode haver uma quantidade elevada de instâncias de *bad smells* e, consequentemente, ter uma taxa alta de recuperações de instâncias é desejável. Em geral, a concentração de valores de *Recall* foi alta. Portanto, pode-se concluir que a participação do especialista foi positiva no sentido de que ele ajudou a melhorar a qualidade das listas de referência providas pela ferramenta *JSpIRIT*.

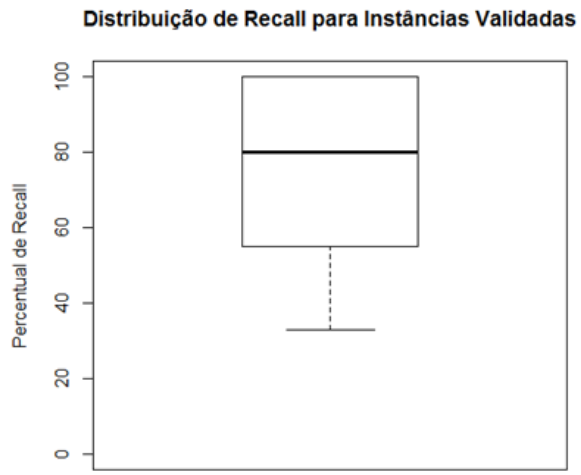

Figura 6.8. Distribuição de *Recall* para instâncias avaliadas pelo especialista

Tabela 6.9. Tabela de análise descritiva de *Recall* Considerando todos os *Bad Smells* avaliados pelo especialista

Percentil	**0%**	**25%**	**50%**	**75%**	**100%**
Recall	33,0	55,0	80,0	100,0	100,0

A Figura 6.9 apresenta a distribuição de Precisão considerando todos os *bad smells* cujas instâncias foram validadas pelo especialista, em relação aos dois sistemas avaliados. A Tabela 6.10 provê a análise descritiva em termos dos percentis do *boxplot*.

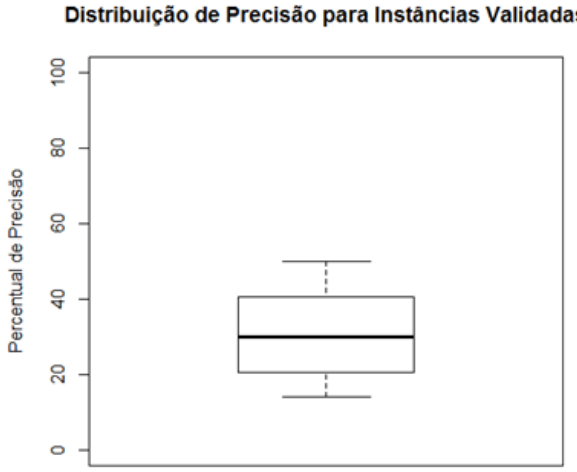

Figura 6.9. Distribuição de Precisão para instâncias avaliadas pelo especialista

Tabela 6.10. Tabela de análise descritiva de precisão para todos os *Bad Smells* avaliados pelo especialista

Percentil	**0%**	**25%**	**50%**	**75%**	**100%**
Precisão	14,0	20,5	30,0	40,5	50,0

Ao contrário das observações feitas em relação a *Recall*, a Figura 6.9 indica uma mediana de Precisão de 30%, um valor baixo porém esperado se verificada a concentração alta de *Recall* apresentada na Figura 6.8. Além disso, nota-se que o maior valor de Precisão observado é 50%. Isso indica que, no melhor caso, dados os resultados de uma estratégia de detecção, metade dos resultados providos foram válidos. Esses resultados sugerem ou que o especialista, por vezes, não foi capaz de identificar manualmente algumas instâncias válidas de *bad smells* ou que as estratégias propostas indicam mais instâncias de *bad smells* do que realmente existe. Avalia-se, aqui, que

detectar *bad smells* manualmente pode ser uma tarefa complexa, e determinar se um trecho de código é anômalo é subjetivo. Por exemplo, o especialista reportou que a detecção de *Feature Envy* demandou uma examinação minuciosa do código-fonte, em relação a chamadas de métodos, manipulação de atributos e outros aspectos dos sistemas analisados.

6.2 Análise com a Ferramenta *JDeodorant*

Os resultados obtidos pelas estratégias propostas nesta dissertação também foram comparados com os resultados obtidos pela *JDeodorant* em termos de *Recall*, Precisão e *F-measure*.

6.2.1 Análise de *Recall*

Análise de *Recall* Por *Bad Smell*. A Figura 6.10 apresenta a distribuição de *Recall* por *bad smell*, levando em consideração os resultados de detecção providos pela *JDeodorant* e pelas estratégias de detecção apresentadas nesse estudo. É importante relembrar que esta ferramenta não se propõe a identificar *Data Class* e *Refused Bequest*. Portanto, ambos os *bad smells* foram descartados nessa análise. A Tabela 6.11 provê a análise descritiva em termos de percentis de cada *boxplot*.

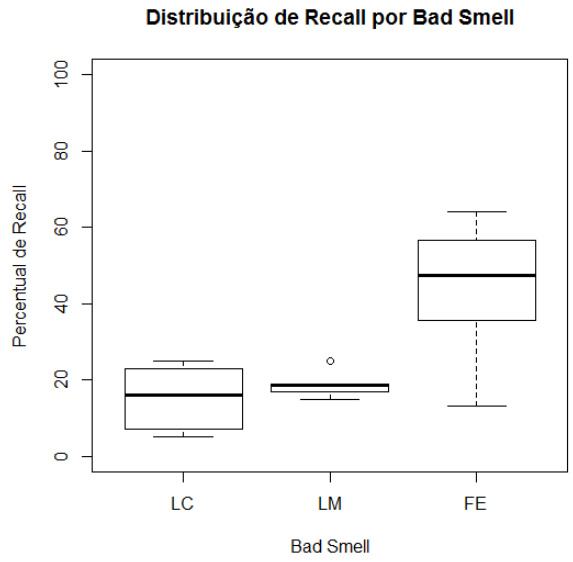

Figura 6.10. Distribuição de *Recall* por *Bad Smell* para *JDeodorant*

Tabela 6.11. Tabela de análise descritiva de *Recall* por *Bad Smell* para *JDeodorant*

Bad Smell	**0%**	**25%**	**50%**	**75%**	**100%**
Large Class	5,0	7,5	16,0	22,5	25,0
Long Method	15,0	17,25	18,5	19,0	25,0
Feature Envy	13,0	35,75	47,5	56,25	64,0

Em relação aos *bad smells Large Class e Long Method*, observa-se uma concentração baixa dos valores de *Recall* para as estratégias de detecção propostas em relação à ferramenta *JDeodorant*. Porém, ao verificar as listas de referência observou-se que a quantidade de resultados de detecção reportados pela ferramenta é, em geral, muito maior do que a quantidade identificada pelas estratégias. Por exemplo, tomando como base o sistema *Checkstyle*, a ferramenta identificou: (i) 81 instâncias de *Large Class*, contra 4 instâncias reportadas pela respectiva estratégia de detecção proposta isto é, 95% mais instâncias; e (ii) 110 instâncias de *Long Method*, contra 40 instâncias reportadas pela estratégia de detecção desse *bad smell* ou seja, 60% mais instâncias. Além disso, para o sistema *JsXe*: (i) 29 instâncias de *Large Class*, contra 4 instâncias reportadas pela respectiva estratégia de detecção isto é, 86% mais instâncias; e (ii) 126 instâncias de *Long Method*, contra 52 instâncias reportadas pela estratégia de detecção desse *bad smell* ou seja, 59% mais instâncias.

Em contrapartida, a concentração dos valores de *Recall* para *Feature Envy* são, em geral, moderados, com mediana próxima a 50%. O maior valor de *Recall* obtido foi em torno de 65%. Além disso, o primeiro quartil está próximo de 40%, o que significa que, para $\frac{3}{4}$ dos sistemas avaliados, o *Recall* foi significativo, menor ou igual a 40%. Note-se que, ao contrário do que foi observado na análise com *JSpIRIT*, o *Recall* para esse *bad smell* foi, de certa forma, significativo. A ferramenta *JDeodorant* possui uma abordagem híbrida de detecção de *bad smells*. De acordo Fernandes et al. [2016], a detecção de *bad smells* implementada pela ferramenta *JDeodorant* é baseada em métricas de software e *Abstract Syntax Tree (AST)*. Tal abordagem híbrida pode ter provido resultados de detecção mais alinhados aos resultados reportados pela estratégia proposta quando comparado à *JSpIRIT*.

Análise de *Recall* Considerando Todos os *Bad Smells*. A Figura 6.11 apresenta a distribuição de *Recall* para todos os três *bad smells* identificados pela *JDeodorant*, em relação aos resultados de detecção apresentados pelas estratégias propostas. A concentração de valores de *Recall* é, em geral, baixa, ainda que os resultados obtidos isoladamente para *Feature Envy* tenham sido significativos. Isso se deve ao fato de

que, para os outros dois *bad smells*, ou seja, *Large Class e Long Method*, os valores de *Recall* foram baixos o suficiente para tender a distribuição geral à mediana de 20%, aproximadamente. Esses resultados indicam que a taxa de detecção de *bad smells* apresentada pelas estratégias propostas, em relação à ferramenta *JDeodorant*, foi pouco expressiva. A Tabela 6.12 provê a análise descritiva em termos dos percentis do *boxplot*

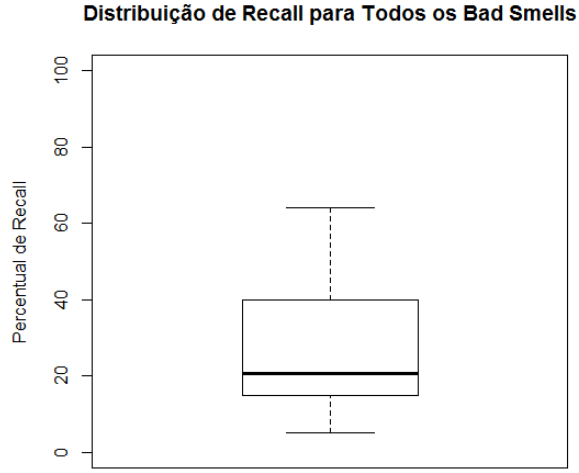

Figura 6.11. Distribuição de *Recall* Considerando Todos os *Bad Smells* para *JDeodorant*

Tabela 6.12. Análise descritiva de *Recall* considerando Todos os *Bad Smells* para *JDeodorant*

Percentil	**0%**	**25%**	**50%**	**75%**	**100%**
Recall	5,0	15,5	20,5	39,0	64,0

6.2.2 Análise de Precisão

Análise de Precisão por *Bad Smell*. A Figura 6.12 apresenta a distribuição de Precisão por *bad smell*, considerando os resultados de detecção providos pela *JDeodorant* e pelas estratégias de detecção propostas nesta dissertação. Tabela 6.13 provê a análise descritiva em termos de percentis de cada *boxplot*.

Para os *bad smells Large Class e Long Method*, obteve-se uma significativa distribuição dos valores de Precisão para as estratégias propostas, em relação à *JDeodorant*. As medianas estão em torno de 50% e 90%, respectivamente. No caso de *Large Class*, 50% dos sistemas avaliados possuem mais de 90% de Precisão, chegando ao máximo

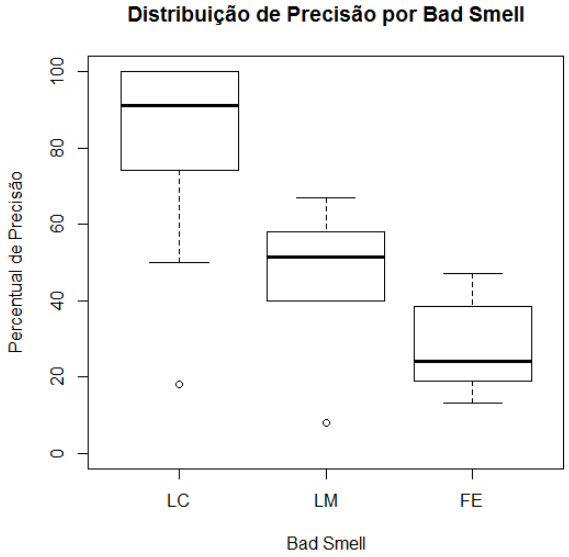

Figura 6.12. Distribuição de Precisão por *Bad Smell* para *JDeodorant*

Tabela 6.13. Tabela de análise descritiva de Precisão por *Bad Smell* para *JDeodorant*

Bad Smell	**0%**	**25%**	**50%**	**75%**	**100%**
Large Class	18,0	74,5	91,0	100,0	100,0
Long Method	8,0	42,5	51,5	56,75	67,0
Feature Envy	13,0	19,5	24,0	36,75	47,0

de 100%. Além disso, para *Long Method*, observa-se uma coerência entre os valores obtidos, que estão entre 40% e 70%. Em outras palavras, a precisão apresentou pouca discrepância de valores entre os sistemas avaliados, no caso de *Long Method*. Assim, observa-se que, embora *Recall* tenha sido baixo para ambos os *bad smells*, a Precisão é grande. Ou seja, embora a ferramenta tenha retornado uma quantidade baixa de instâncias em relação às estratégias, o resultado das estratégias foi, em geral, preciso.

Em relação a *Feature Envy*, observa-se uma distribuição baixa dos valores de precisão, com mediana próxima a 20%. Isso vai de encontro aos resultados significativos de *Recall*. Uma possível justificativa é que, quanto maior a quantidade de resultados recuperados por uma estratégia de detecção, maiores as chances de ocorrência de falsos positivos, conforme observado no experimento com *JSpIRIT*.

Análise de Precisão Considerando Todos os *Bad Smells*. A Figura 6.13 apresenta a distribuição de precisão para todos os três *bad smells* identificados pela *JDeodorant*, em relação aos resultados de detecção apresentados pelas estratégias propostas. Em

geral, os valores obtidos foram altos, com mediana próxima a 50%. Os resultados obtidos individualmente para *Feature Envy* foram baixos, mas não o suficiente para enviesar a distribuição geral, uma vez que *Large Class e Long Method* apresentaram concentração significativamente alta de precisão. Portanto, tem-se que a acurácia das estratégias, de forma geral, foi expressiva em relação à *JDeodorant*. A Tabela 6.14 provê a análise descritiva em termos dos percentis do *boxplot*.

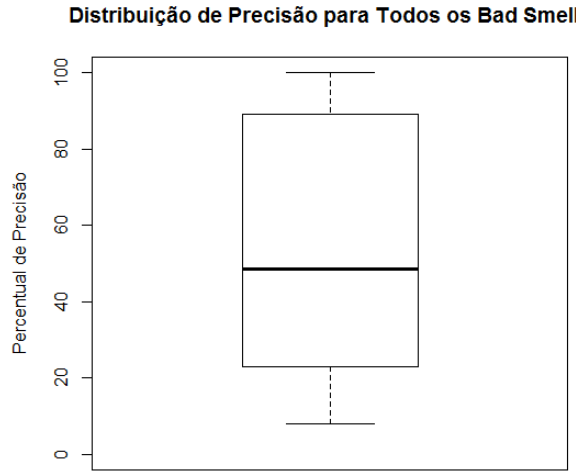

Figura 6.13. Distribuição de Precisão para todos os *Bad Smells* para *JDeodorant*

Tabela 6.14. Análise descritiva de Precisão considerando todos os *Bad Smells* para *JDeodorant*

Percentil	**0%**	**25%**	**50%**	**75%**	**100%**
Precisão	8,0	23,5	48,5	85,5	100,0

6.2.3 Análise de *F-measure*

Análise de *F-measure* por *Bad Smell*. A Figura 6.14 apresenta a distribuição de *F-measure* obtida por *bad smell*. Nela, são apresentados os resultados de distribuição para *Large Class (LC), Long Method (LM) e Feature Envy (FE)*. A Tabela 6.15 provê a análise descritiva em termos dos percentis de cada *boxplot*. Uma discussão acerca dos resultados mostrados na Figura 6.14 é apresentada a seguir.

De acordo com a Figura 6.14, tem-se que nenhum dos *bad smells* analisados pela *JDeodorant* obteve *F-measure* maior que 50%. Além disso, as medianas obtidas

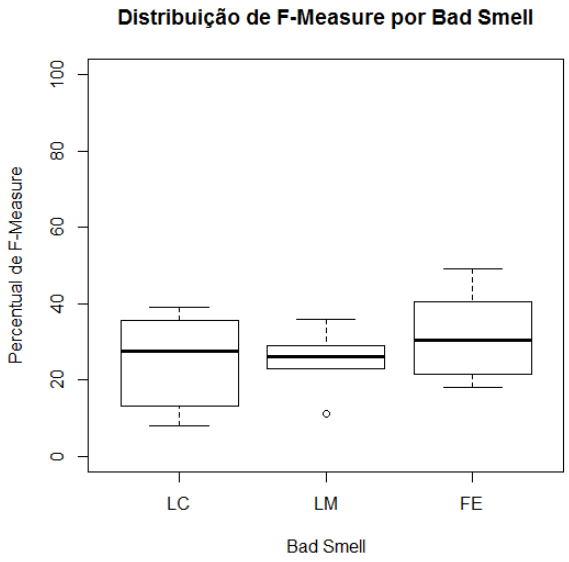

Figura 6.14. Distribuição de *F-measure* por *Bad Smell* para *JDeodorant*

Tabela 6.15. Tabela de análise descritiva de *F-measure* por *Bad Smell* para *JDeodorant*

Bad Smell	**0%**	**25%**	**50%**	**75%**	**100%**
Large Class	8,0	14,0	27,5	34,8	39,0
Long Method	11,0	23,3	26,0	28,8	36,0
Feature Envy	18,0	21,8	30,5	39,3	49,0

não ultrapassaram 30%. Isso significa que, balanceando-se *Recall* e Precisão, obteve-se uma acurácia baixa. Portanto, para nenhum dos três *bad smells* avaliados: *Large Class, Long Method e Feature Envy*, as estratégias de detecção propostas identificaram resultados próximos aos da *JDeodorant*.

Um dos fatores que podem justificar a discordância entre os resultados providos pela *JDeodorant* e pelas estratégias de detecção propostas nesta dissertação é a abordagem híbrida utilizada pela *JDeodorant* para detecção de *bad smells*.

Outro fator que pode ter impactado a análise é a diferença entre a quantidade de resultados retornados pela *JDeodorant* em relação às estratégias propostas. Por exemplo, para o sistema *JEdit*, a ferramenta identificou 172 instâncias de *Large Class*, enquanto que a estratégia proposta retornou 34 instâncias, uma diferença de aproximadamente 80%. Para o sistema *Squirrel SQL*, obteve-se 21 instâncias de *Large Class* providas pela *JDeodorant* enquanto apenas 5 instâncias foram identificadas pela estratégia de detecção. Além disso, 40 instâncias de *Long Method* foram retornadas pela ferramenta, ao passo que somente 12 instâncias foram providas pela estratégia.

Análise de *F-measure* Considerando Todos os *Bad Smells* para *JDeodorant*. A Figura 6.15 apresenta a distribuição de *F-measure* em relação a todos os *bad smells* analisados: *Large Class (LC), Long Method (LM) e Feature Envy (FE)*. A Tabela 6.16 provê a análise descritiva em termos dos percentis do *boxplot*.

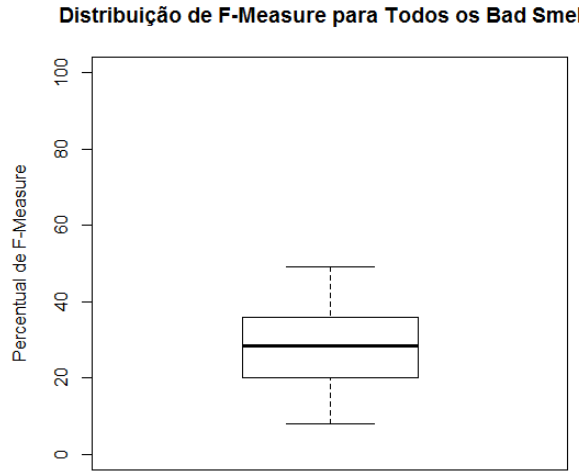

Figura 6.15. Distribuição de *F-measure* para todos os *Bad Smells* para *JDeodorant*

Tabela 6.16. Análise descritiva de *F-measure* considerando todos os *Bad Smells* para *JDeodorant*

Percentil	**0%**	**25%**	**50%**	**75%**	**100%**
F-measure	8,0	20,25	28,50	35,50	49,0

Em geral, observa-se uma concentração baixa dos valores de *F-measure*, com mediana de aproximadamente 30%, provavelmente ocasionada pelas baixas taxas de *Recall* para *Large Class e Long Method*, bem como os valores baixos de precisão obtidos para *Feature Envy*. O maior valor obtido de *F-measure* foi 49%. Esse resultado indica que, no geral, os resultados das estratégias de detecção propostas, quando analisadas como um todo, têm baixa coincidência com os resultados de *JDeodorant*.

6.3 Ameaças à Validade dos Experimentos

A seguir, discutem-se os quatro tipos de ameaça à validade desse experimento, de acordo com as diretrizes de Wohlin et al. [2012]. São elas: ameaças de construção,

ameaças internas, ameaças externas e ameaças de conclusão.

Ameaças de construção. Para esse experimento foram selecionados de forma não-sistemática sistemas de software do *Qualitas.class Corpus 2013*. Esta escolha foi baseada na capacidade de analisar tais sistemas por meio das ferramentas *JSpIRIT* e *JDeodorant*, que mostraram-se não-escaláveis para sistemas acima de 60MB e, na disponibilidade do código-fonte dos sistemas para *download*. O sistema *Sandmark 3.4* por exemplo, não continha o código-fonte dentro do arquivo disponibilizado. Os sistemas foram selecionados aleatoriamente dada a grande quantidade disponível pelo *Qualitas.class Corpus 2013* e, portanto, alguns sistemas passíveis de análise foram descartados. Esse procedimento pode ter impactado negativamente no cálculo de *Recall*, Precisão e, consequentemente, *F-measure*, pois quanto maior o tamanho da amostra, maior tende a ser sua representatividade e diversidade [Nagappan et al., 2013].

Além disso, as estratégias de detecção deste estudo foram propostas de acordo com as métricas de software disponíveis tanto no catálogo de Filó [2014], quanto no *Qualitas.class Corpus 2013*. Portanto, a carência de métricas pode ter impactado negativamente nos resultados obtidos, uma vez que as estratégias podem não ter considerado alguns aspectos que caracterizam cada tipo de *bad smell*. A fim de amenizar esse problema, selecionou-se cuidadosamente cada métrica para composição das estratégias, buscando-se utilizar métricas apropriadas.

O catálogo de valores referência selecionado Filó et al. [2015] também impacta diretamente nos resultados obtidos, pois tais valores são usados para composição das estratégias. Para mitigar tal impacto, as faixas de valores referência foram definidas cuidadosamente de acordo com as características de cada *bad smell*.

Além disso, a qualidade dos resultados de detecção providos pelas *JSpIRIT e JDeodorant* não é garantida, pois a literatura tem reportado taxas insuficientes de *Recall* e Precisão no caso de detecção automatizada de *bad smells* por essas ferramentas [Bellon et al., 2007; Fernandes et al., 2016]. Portanto, utilizá-los como referência irrefutável de instâncias de *bad smells* em sistemas de software pode acarretar em uma análise com baixa confiabilidade. Porém, isso não foi, necessariamente um problema para esse estudo, pois os resultados providos pelas *JSpIRIT e JDeodorant* não foram considerados como totalmente verdadeiros, e sim como uma lista de referência para fins de comparação com os resultados gerados pelas estratégias de detecção propostas.

No caso do *Long Method,* a ferramenta *JDeodorant* não identificou instâncias desse *bad smell* para alguns dos sistemas avaliados. Isso pode ter sido ocasionado por (i) limite excedido de memória ou (ii) alto grau de restrição da abordagem utilizada pela ferramenta para detectar esse tipo de *bad smell*.

Ameaças internas. Parte da coleta dos dados para análise possui algumas ameaças relacionadas a fatores humanos, pois os resultados providos pelas *JSpIRIT e JDeodorant* foram transferidos manualmente para planilhas para viabilização do cálculo de *Recall*, Precisão e *F-measure*. Além disso, a aplicação das estratégias propostas foram realizadas por meio de uma ferramenta, *FindSmells*, que pode conter erros. A fim de mitigar esse problema, tanto o código-fonte da *FindSmells* quanto as planilhas para cálculo de *Recall*, Precisão e *F-measure* foram validados por um especialista em programação orientada por objetos. Além disso, foi efetuada uma verificação manual dos resultados para identificar a interseção entre os resultados obtidos pela *JSpIRIT* e pela *JDeodorant* com os resultados identificados pelas estratégias de detecção propostas. Esses resultados também foram conferidos por um especialista como forma de tratamento para essa ameaça.

Para a seleção dos 12 sistemas avaliados neste estudo, foi necessário importar o projeto de código-fonte de cada sistema na ferramenta de desenvolvimento *Eclipse IDE*. Isso foi necessário, pois tanto a *JSpIRIT* quanto a *JDeodorant* são *plugins* do *Eclipse* e por isso requerem que os sistemas de entrada sejam importados para identificação de *bad smells*. Porém, nem sempre foi possível importar os sistemas, como por exemplo, nos casos dos sistemas *ANTLR, ArgoUML e Tomcat* que apresentaram erro. Eventualmente, a falta de uma configuração de importação adequada pode ter ocasionado no descarte de sistemas que poderiam ter sido utilizados no estudo. Como tratamento desta ameaça, vários sistemas foram submetidos para importação a fim de obter-se uma quantidade suficiente de sistemas para análise.

Por fim, foi observado que o *Qualitas.class Corpus 2013* possui sistemas com mais de um arquivo XML de métricas de software. Uma vez que não estava claro na documentação do *Corpus* se os arquivos correspondem a diferentes pacotes ou versões distintas de um mesmo sistema, esses sistemas foram removidos da análise, por precaução. Essa decisão foi tomada para evitar possíveis conflitos entre o conteúdo dos arquivos, como por exemplo, valores diferentes para uma mesma métrica.

Ameaças externas. Baseado em trabalhos anteriores que investigam *bad smells* em código-fonte [Bellon et al., 2007; Filó, 2014; Filó et al., 2015; Fernandes et al., 2016], foram escolhidas as métricas *Recall*, Precisão e *F-measure* para análise da efetividade dos valores referência utilizados por meio das estratégias de detecção propostas. Ainda que outras medições possam ser interessantes neste contexto de estudo, as medições escolhidas mostraram-se eficazes para apoiar a discussão apresentada nesta dissertação. Além disso, para validação da análise realizada, contou-se com o apoio de um especialista em programação orientada por objetos e com conhecimento sobre os *bad*

smells de Fowler [1999].

Ameaças de conclusão. Neste estudo, foi avaliado um conjunto de 12 sistemas de software distintos do *Qualitas.class Corpus 2013*. Dos 111 sistemas disponíveis no *Corpus*, tentou-se importar 58 sistemas, porém 46 foram descartados pelos seguintes motivos: (i) problemas na importação do código-fonte na *Eclipse IDE*; ou (ii) limite excedido de memória pela *Eclipse IDE*; ou (iii) existência de mais de um arquivo XML com métricas de software para o mesmo sistema.

Por questões de escalabilidade das ferramentas *JSpIRIT e JDeodorant*, o maior sistema avaliado, em termos de espaço disco, possui no máximo 60 MB, 1 KNOC (classes) e 7 KNOM (métodos). Ainda que, sob o ponto de vista da autora dessa dissertação, os 12 sistemas escolhidos foram suficientes para a análise proposta neste estudo, esse conjunto de sistemas pode não refletir a realidade de desenvolvimento no contexto de sistemas de grande porte. Além disso, todos os sistemas disponibilizados no *Qualitas.class Corpus 2013* são *open-source* e implementados na linguagem de programação Java. Isso também pode dificultar a generalização dos resultados obtidos para outros contextos de desenvolvimento, como por exemplo, para outras linguagens de programação.

Devido a limitações de tecnologia, como é o caso dos problemas de memória excedida que aconteceram ao tentar executar sistemas de grande porte, a análise de mais sistemas de software foi inviável. Como tratamento desta ameaça, buscou-se selecionar sistemas conhecidos pela comunidade acadêmica, investigados na literatura e com quantidades significativas de instâncias de *bad smells*, como por exemplo, *AOI, JEdit e Webmail* [Sales et al., 2013; Cardoso & Figueiredo, 2015].

6.4 Lições Aprendidas

Analisando-se os resultados dos experimentos, que comparou as estratégias de detecção propostas em relação as ferramentas de detecção automatizada de *bad smells*, *JSpIRIT e JDeodorant*, constatou-se uma relação entre *Recall* e Precisão para um mesmo tipo de *bad smell*. Especificamente, no experimento com a *JSpIRIT*, o *bad smell Long Method* obteve *Recall* médio de aproximadamente 80% e, por outro lado, obteve uma Precisão média em torno de 35%. Observa-se que, para alguns *bad smells* avaliados, quanto maior o *Recall*, menor a Precisão. Esse resultado corresponde a uma quantidade menor de ocorrência de Falsos Negativos e uma quantidade maior de ocorrências de Falsos Positivos. Para Padilha et al. [2013], no contexto de detecção de *bad smells*, esse é um bom resultado, pois não detectar um grande número de *bad smells*, Falsos Negativos, é

pior do que se obter um alto número de *bad smells* incorretos, Falsos Positivos, porque os Falsos Positivos podem ser revelados durante a inspeção manual do código.

Também observou-se uma variação substancial de *Recall* de acordo com o *bad smell* detectado pelas estratégias de detecção propostas nessa dissertação. Nesse contexto, observou-se valores muito altos, ou muito baixos, dependendo do tipo de *bad smell*. Para 3 dos 5 *bad smells*, obteve-se mediana maior ou igual a 60%, o que é bastante expressivo dada a complexidade do problema de detectar *bad smells* em termos da grande quantidade de instâncias possíveis [Macia et al., 2012]. Porém, no caso de *Feature Envy*, por exemplo, obteve-se baixos valores de *Recall* e Precisão.

A estratégia de detecção proposta para *Feature Envy* baseia-se em apenas uma métrica, uma vez que não há outras métricas no catálogo de Filó et al. [2015] que foram consideradas indicadores desse *bad smell*. Isso pode ter afetado o desempenho da estratégia de detecção. Todavia os resultados dos estudos realizados não fornecem base para se concluir isso, uma vez que os resultados das ferramentas utilizadas não são considerados como um "oráculo" neste trabalho. A conclusão principal que os resultados desse estudo fornece é que os resultados das estratégias de detecção propostas não são discrepantes em relação as ferramentas disponíveis para detecção de *bad smell*. Além disso, observa-se também que os resultados das estratégias de detecção são mais próximos dos resultados da *JSpIRIT* do que dos da *JDeodorant*.

Capítulo 7

Valores Referência e Detecção Manual de *Bad Smells*

Em geral, detecção automatizada de *bad smells* provida por ferramentas já disponíveis não possui alta acurácia em seus resultados [Bellon et al., 2007; Fernandes et al., 2016]. Por isso, julgou-se necessário avaliar os resultados das estratégias de detecção propostas em relação à análise realizada por um especialista em Orientação por Objetos (OO) e com conhecimento em *bad smells*. O especialista que atuou neste estudo é mestrando em Ciência da Computação na área de Engenharia de Software. Seus trabalhos e seu conhecimento são fortemente voltados para qualidade estrutural de software, em particular, *bad smells*. O objetivo deste estudo é responder a questão de pesquisa *QP1.2* desta dissertação:

QP1.2. *Os valores referência apoiam efetivamente a detecção de bad smells em relação a listas de referência geradas por um especialista com conhecimentos em orientação por objetos e bad smells?*

O sistema utilizado para esse experimento foi o *Apache Maven*. Ele é um dos sistemas que compõem o *Qualitas.class Corpus 2013*. O *Maven* é uma ferramenta para gerenciamento e compreensão de projetos de sistemas de software. Neste estudo, foi avaliada a Versão 3.0.5 dessa ferramenta, que possui 18 MB, 864 classes e 6065 métodos. Esse sistema foi escolhido porque: (i) é conhecido pela comunidade acadêmica, (ii) tem uma quantidade significativa de classes e métodos para análise e possível identificação de *bad smells* e (iii) possui um tamanho viável para análise manual do especialista. Esse estudo foi dividido em duas etapas descritas a seguir.

Etapa 1. Na primeira etapa, o especialista analisou as classes e os métodos do sistema com o objetivo de identificar instâncias dos seguintes *bad smells: Large Class, Long Method, Data Class e Refused Bequest.* A análise de *Feature Envy* demanda uma grande quantidade de tempo. Por essa razão, *Feature Envy* não foi considerado nesse estudo, por limitações de tempo e para não sobrecarregar o especialista que foi voluntário no estudo.

Etapa 2. Na segunda etapa, foi realizada a comparação dos dados obtidos pelo especialista com os dados obtidos pelas estratégias de detecção propostas. As métricas de *Recall*, Precisão e *F-measure* foram utilizadas nesta avaliação.

7.1 Identificação de *Bad Smells* pelo Especialista

O especialista teve uma semana pra conduzir a análise manual do *Maven*. Assim como realizado nos demais experimentos, foi importado o código-fonte do sistema na *Eclipse IDE*. A partir daí, o especialista gerou uma lista de instâncias por *bad smell*.

A fim de compreender os motivos pelos quais tal especialista considerou válidas cada uma das instâncias identificadas, foi solicitado a ele que elencasse os principais critérios levados em consideração durante a identificação de cada tipo de *bad smell*. De modo geral, esses critérios estão alinhados com as definições apresentadas por Fowler [1999]. A seguir, são apresentados os critérios de detecção reportados pelo especialista.

- *Large Class*: classe excessivamente grande, com quantidade exagerada de métodos (responsabilidades) e atributos (conhecimento). Os métodos da classe precisam ser mais complexos do que simples *getters e setters*. Em geral, mais de 1 KLOC foi indicativo de *Large Class*. Comentários foram ignorados sempre que identificados. Interfaces foram ignoradas, quando identificadas.

- *Long Method*: método muito extenso, mas também complexo. Comentários no corpo do método foram ignorados.

- *Data Class*: classe que contém somente construtores, *getters e setters*, basicamente. Caso outros tipos de métodos sejam identificados, eles devem ser considerados de "simples" processamento para que a classe fosse apontada como *Data Class*. O valor referência igual a 4 foi subjetivamente derivado para contagem dos *getters e/ou setters*. Interfaces foram ignoradas, quando identificadas.

- *Refused Bequest*: classe com muitos métodos herdados sobrescritos, e mais uma quantidade considerável de métodos novos. Classes que implementavam interfaces foram ignoradas. Métodos sobrescritos da superclasse abstrata não tiveram peso ao considerar *Refused Bequest*.

7.2 Resultados

A Tabela 7.1 apresenta os resultados de *Recall*, Precisão e *F-measure* para cada tipo de *bad smell* avaliado manualmente pelo especialista no sistema *Maven*. Também foi calculada a média e o desvio padrão desses valores, a fim de apoiar a discussão a seguir.

Tabela 7.1. Resultados do Especialista

Bad Smell	***Recall***	***Precisão***	***F-measure***
	%	%	%
Large Class	70	37	48
Long Method	73	19	30
Data Class	47	32	38
Refused Bequest	100	2	4
Média	72,5	22,5	30
Desvio Padrão	21,7	15,6	18,8

Por meio da análise descrita na Tabela 7.1, obteve-se média de *Recall* de aproximadamente 73%, levando em consideração todos os 4 *bad smells* avaliados. Esse valor indica que as estratégias de detecção, em geral, foram capazes de reportar uma quantidade significativa de instâncias de *bad smells* em relação àquelas identificadas pelo especialista. A média de Precisão obtida foi de aproximadamente 23%, muito aquém do percentual de *Recall* identificado. Isso significa que, embora a quantidade de instâncias recuperadas tenha sido expressiva, a quantidade de falsos positivos é alta, ou seja, a quantidade de instâncias válidas do ponto de vista do especialista é baixa. Consequentemente, essa discrepância afetou negativamente o coeficiente *F-measure* que, ao balancear *Recall e Precisão*, apresentou valor médio de 30%.

7.3 Ameaças à Validade do Experimento

São identificadas as seguintes ameaças à validade deste estudo baseado na literatura [Wohlin et al., 2012].

Ameaças de construção. Para realizar este experimento, utilizou-se o sistema *Maven*, do *Qualitas.class Corpus 2013*. Embora somente um sistema tenha sido analisado, dadas restrições de tempo e disponibilidade do especialista, o *Maven* é um sistema utilizado para estudos na literatura e possui uma quantidade significativa de classes e métodos que podem representar instâncias de *bad smells* [Terra et al., 2013; Tufano et al., 2015]. Porém, a quantidade reduzida de sistemas avaliados pode ter impactado nos resultados de *Recall*, Precisão e *F-measure*. Quanto às estratégias de detecção, foram utilizadas as mesmas do estudo anterior. Dessa forma, as ameaças à validade deste experimento, inerentes a definição das estratégias de detecção, são as mesmas do estudo anterior.

Ameaças internas. Neste estudo, a coleta de dados também possui ameaças relacionadas a fatores humanos, uma vez que a análise do *Maven* foi efetuada manualmente por um especialista em POO e *bad smells* em código-fonte. Tais ameaças podem estar relacionadas à complexidade da tarefa realizada manualmente, bem como às limitações de tempo para execução do estudo. Como forma de tratamento dessa ameaça, disponibilizou-se uma semana para realização da análise, considerando o tamanho do sistema avaliado e a quantidade de *bad smells* a serem identificados: *Large Class, Long Method, Data Class e Refused Bequest*.

O especialista participante deste estudo mostrou ter conhecimento sobre o conceito de valor referência. Nos critérios para detecção dos *bad smells* relatados por ele, observa-se que ele utilizou valores referência arbitrários. Além disso, na definição dos *bad smells Large Classe e Refused Bequest*, o especialista aplicou conceitos mais rigorosos do que aqueles definidos por Fowler [1999]. No caso de *Large Class*, o especialista considerou que, além de a classe possuir muitos métodos, esses métodos precisam ser complexos. Na caso de *Refused Bequest*, o especialista considerou que além de a classe sobrescrever os métodos da superclasse, ela precisa ter também uma quantidade considerável de métodos novos. Esses fatores, portanto, constituem-se em ameaças à validade dos resultados deste estudo.

Ameaças externas. Assim como nos experimentos anteriores, foi escolhido o seguinte conjunto de medições para avaliação dos resultados deste estudo: *Recall*, Precisão e *F-measure*. Portanto, as mesmas ameaças à validade externa presentes no outro estudo, referentes ao cálculo dessas métricas, se aplicam a este estudo.

Ameaças de conclusão. Neste estudo, foi avaliado somente um sistema do *Qualitas.class Corpus 2013*, o *Maven*. Dessa forma, não é possível generalizar os resultados

obtidos para qualquer sistema de software Java existente. Ainda que o *Maven* tenha um expressivo número de classes e métodos, ele não está entre os maiores sistemas do *Qualitas.class Corpus 2013* [Terra et al., 2013]. Porém, este sistema é avaliado em estudos relacionados [Tufano et al., 2015].

7.4 Lições Aprendidas

Uma das principais lições aprendidas com a realização desse experimento foi a diferença entre os critérios adotados pelo especialista na detecção manual de *bad smells*, em comparação com a detecção automatizada utilizando-se ferramentas. Considerando o parecer do especialista quanto à identificação dos 4 *bad smells* avaliados, percebe-se uma maior restritividade de critérios para identificação manual de *bad smells*, em comparação com a detecção automatizada. Além disso, a detecção realizada por ferramentas segue sempre as mesmas regras, pré-definidas, enquanto a detecção manual pode ter um caráter parcialmente subjetivo. Por exemplo, no caso de *Large Class*, o especialista reportou que a quantidade de linhas de código de uma classe é um fator relevante na detecção desse *bad smell*, mas não é um fator decisivo. Assim, foi necessário ele analisar outros aspectos das classes, que podem não ter sido cobertos pelas métricas de software que compõem a respectiva estratégia utilizada pelas ferramentas.

Outra lição aprendida está relacionada à diferença observada entre os resultados de *Recall*, Precisão e *F-measure* computadas neste estudo, com especialista, e nos outros estudos, com ferramentas. A diminuição nos valores de precisão se justifica, pois a análise manual do especialista tende a ser mais restritiva, o que diminui o tamanho das listas de referência. Com isso, as chances de interseção entre os resultados do especialista e das estratégias de detecção propostas tende a diminuir significativamente.

Os resultados deste estudo sugerem que os resultados reportados pelas estratégias de detecção em geral não são discrepantes em relação à análise do especialista. A discrepância grande observada foi na avaliação de *Refused Bequest*. Porém é importante notar que o especialista aplicou uma definição desse *bad smell* que extrapola a definição original de Fowler [1999], na qual a estratégia de detecção proposta é baseada. Os valores de *Recall* foram altos neste estudo, o que indica que as estratégias de detecção propostas identificaram pouca quantidade de falsos negativos em relação aos resultados reportados pelo especialista. Um falso negativo ocorre quando o especialista indica presença do *bad smell*, mas a estratégia de detecção o ignora. Dessa forma, a obtenção de *Recall* alto é importante no contexto de detecção de *bad smell*, uma vez que indica que poucas instâncias de *bad smell* são ignoradas pela estratégia de detecção.

Capítulo 8

Valores Referência e Predição de Falhas

Diversas técnicas são propostas na literatura a fim de apoiar predição de falhas em sistemas de software. Dentre elas, podemos citar aprendizado de máquina [Salfner et al., 2010], análise histórica de sistemas [Graves et al., 2000] e análise estática de código-fonte [Zheng et al., 2006]. Nesse contexto, métricas de software também podem ser utilizadas na predição de falhas. Há vários trabalhos na literatura que correlacionam predição de falhas e métricas de software, tais como: Gyimothy et al. [2005] descrevem como foram calculadas as métricas orientadas por objetos propostas por Chidamber & Kemerer [1994] para ilustrar como a predição de falhas de um código-fonte do navegador *web Mozilla* pode ser feita; Catal & Diri [2009] propõem um agrupamento de valores referência de métricas baseados em abordagens de previsão falhas de software para resolver o problema de prever falhas quando os rótulos de falhas para os módulos estão indisponíveis; e, Couto et al. [2013a] propõem uma abordagem para predição de falhas centrada em evidências de causalidade entre métricas de código-fonte e a ocorrência de defeitos.

Extrair métricas a partir de sistemas de software é uma tarefa relativamente de custo baixo, pois existem ferramentas amplamente utilizadas para este fim tais como: *CodePro Analytix*[1], *Metrics*[2] *e Sonar Qube*[3]. Sendo assim, a utilização de métricas na predição de falhas pode ser mais viável do que a aplicação de técnicas mais complexas como por exemplo, aprendizado de máquina e análise de código-fonte. Portanto, é importante investigar a eficácia dos valores referência para métricas de software nesse

[1]https://marketplace.eclipse.org/content/codepro-analytix
[2]http://metrics.sourceforge.net/
[3]http://www.sonarqube.org/

contexto.

Este capítulo apresenta uma avaliação do catálogo de valores referência proposto por Filó et al. [2015] no contexto de predição de falhas em sistemas de software orientados por objeto. Para tal, foi conduzido um estudo que investiga a seguinte questão de pesquisa:

QP2. *Os valores referência de métricas de software orientados por objetos auxiliam a predizer falhas em um software?*

Com a *QP2*, objetiva-se verificar se as faixas de valores determinadas por Filó et al. [2015] para cada métrica do catálogo apresentado são suficientes indicadores de falhas em sistemas de software. Neste estudo, será verificado se as faixas *Regular/Ocasional* e *Ruim/Raro* do catálogo de Filó et al. [2015] podem ser aplicadas para a predição de falhas em software.

8.1 Escopo do Estudo

Para definir as etapas e execução de um estudo que responda à *QP2*, adotou-se como escopo desse estudo uma metodologia baseada no trabalho de Wohlin et al. [2012].

Analisar os valores referência de métricas de software propostos por Filó et al. [2015]

A fim de verificar a utilidade deles na predição de falhas em sistemas de software

Em relação à proporção de falhas identificadas nas classes cujas medidas correspondem aos intervalos dos valores referência das métricas avaliadas

Sob o ponto de vista de listas de referência de falhas e arquivos de métricas

No contexto de sistemas de código-fonte aberto desenvolvidos em *Java*

8.2 Etapas do Estudo

A Figura 8.1 apresenta as seis etapas definidas para condução do estudo proposto. Essas etapas envolvem desde um mapeamento sistemático da literatura até a análise

dos resultados. Cada etapa é detalhada a seguir.

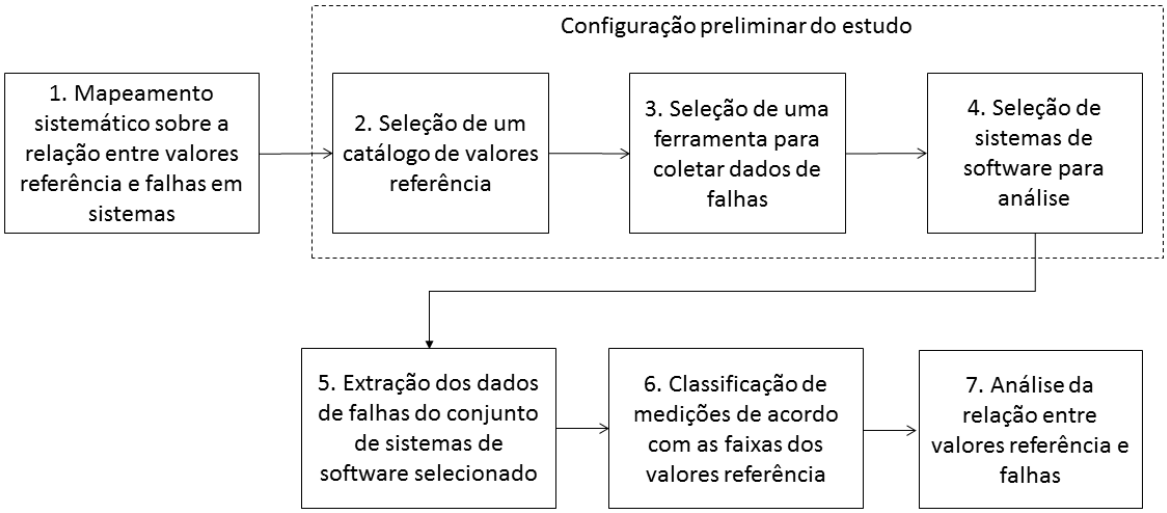

Figura 8.1. Etapas do estudo sobre aplicação dos Valores Referência x Falhas

Etapa 1 – Mapeamento sistemático sobre a relação entre valores referência e falhas. Os trabalhos encontrados por meio do mapeamento sistemático não se preocupam em proporcionar a real utilização de valores referência na indústria de software, provavelmente devido à falta de ferramentas apropriadas e, principalmente, à ausência de valores referência que possam orientar o seu uso em relação à avaliação da qualidade do software. Além disso, percebe-se que alguns trabalhos mais recentes têm se dedicado a definir valores referência apenas para algumas métricas utilizando diferentes abordagens. Uma questão que surge nesse contexto refere-se à eficácia dos valores referência propostos para avaliar os softwares adequadamente. Esse estudo se insere nesse contexto. Ele visa contribuir para preencher uma lacuna importante de conhecimento que se tem a respeito da aplicabilidade de métricas de software e de seus valores referência na prática, mais especificamente, para predizer falhas em sistemas de softwares orientados por objetos.

Etapa 2 – Seleção de um catálogo de valores referência. O catálogo de valores referência proposto por Filó et al. [2015], assim como no estudo anterior, também é utilizado nesse experimento. A descrição sobre esse catálogo está no Capítulo 4 desta dissertação.

Etapa 3 – Seleção de uma ferramenta para coletar dados de falhas. A ferramenta *BugMaps*, proposta por Couto et al. [2013b], foi selecionada para esse

experimento para coletar dados de falhas armazenados em sistemas de *bug-tracking*. A ferramenta provê dados históricos de falhas em sistemas de dois repositórios: *Bugzilla* e *Jira*. A escolha dessa ferramenta se baseou no estudo de Januário & Ferreira [2016], que realizou um estudo comparativo entre ferramentas de coleta de dados de falhas de software. Januário & Ferreira [2016] identificaram que *BugMaps* é a única ferramenta que coleta dados de falhas por cada classe do software, exportando o resultado em formato CSV, que é de fácil manipulação.

Etapa 4 – Seleção de sistemas de software para análise. Para a seleção dos sistemas de software a serem analisados neste estudo, utilizou-se como base o *Qualitas.class Corpus* descrito no Capítulo 4 desta dissertação. Os sistemas foram selecionados considerando a disponibilidade de falhas cadastradas nos repositórios *Bugzilla* e *Jira*. A Tabela 8.1 apresenta a relação dos sistemas selecionados para esse estudo. Ela apresenta o nome e a versão do sistema, o tamanho do sistema e o número de classes analisadas. O número de classes analisadas refere-se a quantidade de classes do sistema associadas as falhas por meio da ferramenta *BugMaps*. O processo para obtenção desses dados é descrito na etapa 5.

Tabela 8.1. Sistemas selecionados do *Qualitas.class Corpus* 2013 para estudo de VR x *falhas*

#	**Sistema**	**Tamanho**	**Nº de Classes Analisadas**
1	ant-1.8.2	37,1 MB	869
2	aspectj-1.6.9	61 MB	1507
3	batik-1.7	94 MB	1473
4	cayenne-3.0.1	7,1 MB	1844
5	derby-10.9.1.0	73 MB	1851
6	jmeter-2.5.1	22 MB	977
7	maven-3.0.5	18 MB	696
8	myfaces-2.1.10	83 MB	1189
9	poi-3.6	49 MB	2149
10	roller-5.0.1	133 MB	500

Etapa 5 – Extração dos dados de falha do conjunto de sistemas de software selecionado. Para coletar os dados de falha dos sistemas selecionados do *Qualitas.class Corpus*, o seguinte processo foi executado [Januário & Ferreira, 2016]: (i) os dados de todas as falhas dos sistemas de software foram coletados nas ferramentas de gerenciamentos de defeitos *Bugzilla e Jira*; isso resultou em um arquivo de formato CSV que é o formato necessário para realizar a entrada dos dados na ferramenta *BugMaps* [Couto et al., 2013b]; (ii) em seguida informou-se à *BugMaps* o período desejado para filtrar as falhas. No caso deste experimento o período fixado compreende

de 2013 a 2016 devido ao fato de às métricas do *Qualitas.class Corpus* serem referentes a 2013 e, (iii) a partir do ID de cada falha presente no repositório dos fontes do projeto, *GitHub*, a *BugMaps* realiza uma associação das falhas com as classes do sistema e exporta os dados para um arquivo em formato CSV.

Etapa 6 – Classificação de medições de acordo com as faixas dos valores referência. Para os sistemas selecionados na Etapa 4 foi realizada uma análise do arquivo de métricas do *Qualitas.class Corpus* referente a cada sistema, a fim de identificar em qual faixa (*Bom, Regular ou Ruim*) o valor de cada métrica, à nível de classe, encontra-se em relação aos valores referência propostos por Filó et al. [2015]. Por meio de *scripts* implementados em *Java*, o valor da métrica foi substituído pelo nome da faixa a qual ele corresponde.

Etapa 7 – Análise da relação entre valores referência e falhas. Nessa etapa considerou-se (i) o arquivo de saída gerado pela ferramenta *BugMaps* com a relação das classes por sistema e suas respectivas falhas e (ii) o arquivo de métricas obtido a partir do *Qualitas.class Corpus* contendo a faixa onde cada valor de métrica se encontra de acordo com o catálogo de Filó et al. [2015]. usando *scripts* os arquivos descritos em (i) e (ii) foram unidos para facilitar a análise da relação entre os valores referência de métricas de software e ocorrência de falhas em cada classe. A análise foi realizada com base em estatística descritiva para investigar a relação entre duas variáveis qualitativas: as faixas que cada valor de métrica se refere e a presença de falhas em cada classe. Para tal, foram usadas tabelas dinâmicas que fazem o cruzamento das variáveis investigadas e calculam a frequência da ocorrência de cada cruzamento. Na sequência, os resultados foram convertidos para porcentagem e foi calculado a média e o desvio padrão.

8.3 Resultados

Nesta seção são apresentados os resultados do estudo que investiga a capacidade de os valores referência de Filó et al. [2015] realizar a predição de falhas em sistemas de software. A Seção 8.3.1 descreve uma análise por métrica de software, baseada nos resultados obtidos para cada um dos 10 sistemas selecionados. A Seção 8.3.2 apresenta uma análise complementar para cada métrica por meio da consolidação dos dados de todos os sistemas.

8.3.1 Análise das Métricas de Software por Sistema

Este estudo investiga para cada uma das 10 métricas de software, no nível de classe, disponíveis nos arquivos do *Qualitas.class Corpus*, a capacidade das faixas de valores determinadas pelos valores referência de Filó et al. [2015], identificarem falhas em sistemas de software. As métricas de software avaliadas nesse estudo são: *DIT, LCOM, NOF, NOM, NORM, NSC, NSF, NSM, SIX e WMC.*

A Tabela 8.2 apresenta por métrica e por sistema, os percentuais de classes *com falhas* classificadas nas faixas *Bom, Regular e Ruim*, dos valores referência de Filó et al. [2015]. O complemento do valor de cada célula, em relação a 100%, corresponde ao percentual de classes que não apresentaram falhas classificados nas respectivas faixas de valores. Por exemplo, no caso de *JMeter*, 83,93% das classes com métrica NOF na faixa *Ruim* apresentaram falhas. Com os resultados exibidos nessa tabela é possível analisar quais faixas de valores foram melhores indicadores da ocorrência de falha por métrica.

Tabela 8.2. Percentual de classes com falhas por sistema para cada métrica de software

Métrica	Faixa	Ant	AspectJ	Batik	Cayenne	Derby	Jmeter	Maven	MyFaces	POI	Roller	Média
		%	%	%	%	%	%	%	%	%	%	%
DIT	*B*om	20,10	9,50	8,46	18,45	25,35	61,56	20,37	0,42	45,41	15,61	**22,52**
	*R*egular	17,35	6,78	14,52	21,54	33,98	42,76	2,25	1,12	27,30	21,70	**18,93**
	*R*uim	8,86	3,85	24,51	3,60	29,58	54,73	0,00	0,00	21,21	0,00	**14,63**
LCOM	*B*om	10,75	4,95	9,56	14,48	18,42	48,52	13,32	0,28	34,45	15,00	**16,97**
	*R*egular	11,54	10,84	9,16	17,84	29,93	70,73	19,35	0,00	44,22	14,89	**22,85**
	*R*uim	33,20	17,14	20,48	30,61	53,02	74,44	34,52	3,41	48,78	20,69	**33,63**
NOF	*B*om	10,17	5,54	10,35	14,77	20,05	49,43	13,82	0,24	36,72	14,33	**17,54**
	*R*egular	22,22	15,38	11,59	25,00	36,79	74,75	28,57	1,22	49,71	21,51	**28,67**
	*R*uim	50,51	22,08	15,09	36,00	54,59	83,93	50,00	6,67	40,00	22,86	**38,17**
NOM	*B*om	9,68	3,25	7,45	12,50	18,93	47,22	14,00	0,27	33,65	14,04	**16,10**
	*R*egular	20,14	8,55	22,04	24,84	24,59	64,15	20,27	0,00	41,50	15,48	**24,16**
	*R*uim	40,24	21,13	18,30	37,78	52,59	79,23	36,36	2,83	55,14	24,47	**36,81**
NORM	*B*om	17,36	7,91	9,59	15,65	25,36	55,96	17,34	0,44	38,46	14,29	**20,24**
	*R*egular	32,00	11,49	27,50	27,38	34,67	63,64	25,00	0,00	51,61	28,81	**30,21**
	*R*uim	22,22	11,76	36,11	60,00	43,64	33,33	0,00	2,70	80,00	40,00	**32,98**
NSC	*B*om	18,61	7,81	10,43	15,38	27,85	57,64	17,24	0,63	38,63	16,82	**21,10**
	*R*egular	16,25	9,46	13,77	24,27	25,16	48,57	18,33	0,00	45,65	11,11	**21,26**
	*R*uim	11,54	16,36	10,98	28,72	23,68	36,59	12,50	0,00	34,09	8,33	**18,28**
NSF	*B*om	11,97	6,09	10,72	16,33	24,50	41,26	16,37	0,41	38,26	12,89	**17,88**
	*R*egular	29,13	15,31	10,29	25,77	37,70	67,14	36,36	0,56	37,16	28,21	**28,76**
	*R*uim	41,57	21,43	14,67	21,62	38,60	83,96	33,33	1,85	47,62	50,00	**35,47**
NSM	*B*om	15,45	7,24	10,40	17,28	25,35	52,33	15,91	0,47	34,90	15,17	**19,45**
	*R*egular	32,14	14,29	24,07	7,89	40,79	92,86	25,00	1,82	57,14	18,18	**31,42**
	*R*uim	51,16	17,65	8,77	34,78	46,67	88,89	52,94	0,00	70,97	37,50	**40,93**
SIX	*B*om	16,38	6,65	7,40	14,83	25,38	54,15	18,20	0,39	41,85	12,87	**19,81**
	*R*egular	21,55	13,46	17,48	20,28	32,08	68,22	13,64	1,52	33,54	22,94	**24,47**
	*R*uim	9,52	1,92	34,25	25,00	27,44	30,77	0,00	0,00	25,00	33,33	**18,72**
WMC	*B*om	7,07	1,87	5,04	9,87	12,21	38,71	11,28	0,00	27,24	7,37	**12,06**
	*R*egular	15,73	7,35	16,58	21,27	26,61	70,27	23,36	0,45	45,00	17,24	**24,39**
	*R*uim	44,25	23,40	22,60	47,62	56,35	91,21	46,30	3,17	71,23	39,24	**44,54**

DIT. Para 5 dos 10 (50%) sistemas avaliados, a métrica DIT obteve o maior percentual de classes com falhas na faixa *Bom*, com aproximadamente 23% das classes nessa faixa. Na faixa *Regular*, 4 dos 10 (40%) sistemas obtiveram os maiores percentuais de classes com falhas, média de 19% das classes, contra 1 (10%) sistema na faixa *Ruim*. Observa-se que somando o resultado das faixas *Regular* e *Ruim*, que são as mais críticas de acordo com Filó et al. [2015], tem-se um resultado equivalente ao obtido na faixa *Bom*. Segundo Filó et al. [2015], a faixa *Bom* representa os valores mais frequentes que possivelmente indicam boas práticas de programação. Porém esse resultado sugere que os valores referência de DIT não podem ser considerados como bons indicadores de falhas em software.

LCOM. Para a métrica LCOM nenhum dos 10 sistemas avaliados obteve o maior percentual de classes com falhas nas faixas *Bom* ou *Regular*. Em média, ambas as faixas apresentaram respectivamente 14% e 18% de classes com falha. Em contrapartida, 100% dos sistemas obtiveram o maior percentual de classes com falhas na faixa *Ruim*. O percentual médio corresponde a, aproximadamente, 29% das classes. Esse valor é até 15% maior do que o obtido nas faixas anteriores. Portanto a faixa *Ruim* para a métrica LCOM mostrou-se eficaz na indicação de ocorrência de falhas em sistemas de software. Os resultados sugerem que os valores referência de LCOM são úteis à predição de falhas.

NOF. Não foi obtido para a métrica NOF, em nenhum dos 10 sistemas avaliados, o maior percentual de classes com falhas na faixa *Bom*. Em média, esta faixa apresentou 18% de classes com falha, aproximadamente. Para as faixas *Regular* e *Ruim*, respectivamente, 1 (10%) e 9 (90%) dos 10 sistemas apresentaram o maior percentual de classes com falhas. Os percentuais médios correspondem a, aproximadamente, 29% para a faixa *Regular* e 38% para a faixa *Ruim*. Note que os percentuais de classe por faixa aumentam conforme aumenta a criticidade das faixas, de *Bom* a *Ruim*. Portanto, a faixa *Ruim* para NOF mostrou-se eficaz na indicação de falhas em sistemas e os valores referência de NOF mostraram-se úteis à predição de falhas.

NOM. Dos 10 sistemas avaliados, nenhum deles apresentou o maior percentual de classes com falhas na faixa *Bom* para a métrica NOM. O percentual médio de classes com falhas nessa faixa é de 16%, aproximadamente. Em contrapartida, para as faixas *Regular* e *Ruim*, respectivamente, 1 (10%) e 9 (90%) dos 10 sistemas apresentaram o maior percentual de classes com falhas. Os percentuais médios correspondem a, aproximadamente, 24% para a faixa *Regular* e 37% para a faixa *Ruim*. Também neste

caso, os percentuais de classe por faixa aumentam conforme aumenta a criticidade das faixas. Disso, conclui-se que a faixa *Ruim* para NOM foi eficaz na indicação de falhas. Os valores referência de NOM mostraram-se então, úteis à predição de falhas em software.

NORM. Nenhum dos 10 sistemas avaliados apresentou o maior percentual de classes com falhas na faixa *Bom* para a métrica NORM. O percentual médio de classes com falhas nessa faixa é de 20%, aproximadamente. Porém, para as faixas *Regular* e *Ruim*, respectivamente, 3 (30%) e 7 (70%) dos 10 sistemas apresentaram o maior percentual de classes com falhas. Em média, tais faixas obtiveram, aproximadamente, 30% para a faixa *Regular* e 33% para a faixa *Ruim*. Assim como nas análises de NOF e NOM, observa-se que, quanto mais crítica a faixa, maior o percentual de classe com falhas. Pode-se concluir que a faixa *Ruim* para NORM é um bom indicador de falhas em sistemas. Dessa forma, os valores referência de NORM mostram-se úteis à predição de falhas em software.

NSC. Considerando-se a faixa *Bom* da métrica NSC, obteve-se o maior percentual de classes com falhas para 5 dos 10 (50%) sistemas avaliados. Isso corresponde, em média, a aproximadamente 21% das classes nessa faixa. Entretanto, em relação a faixa *Regular*, 2 dos 10 (20%) sistemas obtiveram os maiores percentuais de classes com falhas, média de 19% das classes, enquanto que 3 (30%) sistemas possuem os maiores percentuais na faixa *Ruim*. Novamente, somando-se o resultado das faixas mais críticas, isto é, *Regular* e *Ruim*, o total de sistemas equivale ao resultado da faixa *Bom*. Diante disso, conclui-se que os valores referência de Filó et al. [2015] para NSC não são bons indicadores de falhas em sistemas, apesar de o catálogo de Filó et al. [2015] apresentar a faixa Bom dessa métrica como indicador de boas práticas de programação.

NSF. Para a métrica NSF, nenhum dos 10 sistemas avaliados obteve o maior percentual de classes com falhas na faixa *Bom*. Nesse caso, o percentual médio de classes com falhas é de 18%, aproximadamente. Contudo, em relação as faixas *Regular* e *Ruim*, respectivamente, 2 (20%) e 8 (80%) dos 10 sistemas apresentaram o maior percentual de classes com falhas. Os respectivos percentuais médios de classes com falhas são, aproximadamente, 29% para a faixa *Regular* e 35% para a faixa *Ruim*. Mais uma vez, observa-se que, quanto maior a criticidade da faixa, maior o percentual de classes com falhas. Portanto, os dados sugerem que os valores referência de NSF são úteis à predição de falhas em sistemas.

NSM. Em relação a métrica NSM, a faixa *Bom* não obteve o maior percentual de classes com falhas para qualquer dos 10 sistemas avaliados. O percentual médio de classes com falhas nessa faixa é de 19%, aproximadamente. Entretanto, em relação as faixas *Regular* e *Ruim*, respectivamente, 2 (20%) e 8 (80%) dos 10 sistemas apresentaram o maior percentual de classes com falhas. Em média, os percentuais de classes com falhas foram, aproximadamente, 31% para a faixa *Regular* e 41% para a faixa *Ruim*. Note que o percentual de classes cresce de acordo com a severidade das faixas. Assim, conclui-se que os valores referência de NSM são capazes de indicar falhas em sistemas.

SIX. Para a métrica SIX, 2 dos 10 (20%) sistemas apresentaram o maior percentual de classes com falhas na faixa *Bom*. Em média, obteve-se 20% de classes com falhas nessa faixa, aproximadamente. Em relação as faixas *Regular* e *Ruim*, respectivamente, 5 (20%) e 3 (80%) dos 10 sistemas apresentaram o maior percentual de classes com falhas. Seus respectivos percentuais médios de classes com falhas foram, aproximadamente, 24% para a faixa *Regular* e 19% para a faixa *Ruim*. Note que houve um crescimento no resultado de *Bom* em relação a *Regular*, mas um decrescimento de *Regular* para *Ruim*. Logo, para SIX, a faixa *Regular* foi a mais eficaz na indicação de falhas em sistemas. Neste caso, também, pode-se considerar que a métrica SIX é útil na predição de falhas de software.

WMC. Nenhum dos 10 sistemas avaliados apresentou, para a métrica WMC, o maior percentual de classes com falhas na faixa *Bom* ou *Regular*. O percentual médio aproximado de classes com falha nessas faixas foi de 12% e 24%, respectivamente. Assim, observa-se que, para 100% dos sistemas, o maior percentual de classes com falhas foi classificado na faixa *Ruim*, com percentual médio de aproximadamente 45% de classes. Esse valor corresponde a até 33% maior do que o obtido nas faixas anteriores. Portanto, a faixa *Ruim* para WMC foi capaz de indicar a maior quantidade de falhas em sistemas. Os resultados sugerem, então, que WMC é útil na predição de falhas de software.

O objetivo deste estudo foi avaliar a capacidade dos valores referência de Filó et al. [2015] predizer falhas em sistemas de software. Para este fim, foram coletados os dados das métricas para 10 sistemas do *Qualitas.class Corpus* Versão 2013, ao passo que falhas referentes a esses mesmos sistemas foram coletadas a partir de versões desenvolvidas entre 2013, ano da coleta das métricas e 2016, ano da conclusão deste trabalho. Observa-se que, para 7 das 10 métricas de software analisadas nesse estudo, a faixa *Ruim* mostrou-se eficaz para indicar falhas. No caso da métrica SIX, observa-se que a faixa *Regular* foi a mais eficaz na indicação de falhas. Para as métricas DIT e

NSC, não foi possível afirmar que os valores referência são indicadores de falhas em sistemas. Ressalta-se que DIT e NSC são métricas relacionadas a aplicação de herança. Dessa forma, esse resultado indica que herança não está relacionada à ocorrência de falhas em software. Os resultados deste estudo confirmam que, de forma geral, os valores referência de Filó et al. [2015] podem apoiar a predição de falhas em sistemas de software.

A fim de complementar este estudo, a Seção 8.3.2 provê uma análise geral por métrica independente do sistema avaliado.

8.3.2 Análise por Métrica de Software Independente do Sistema

Este estudo considera todas as classes dos 10 sistemas avaliados, ou seja, a análise é realizada independente do sistema de origem para verificar a relação entre o percentual de classes classificadas nas faixas: *Bom, Regular e Ruim*, determinadas por Filó et al. [2015] e a presença de falhas. A questão de pesquisa *QP2* é respondida considerando-se o conjunto de dados de todos os sistemas selecionados para avaliação.

Para realizar essa análise foi gerado um arquivo consolidando os dados de todos os sistemas. Nesse arquivo foi calculada a frequência de classes com presença e ausência de falhas de cada faixa, *Bom, Regular e Ruim*, por métrica. A Tabela 8.3 apresenta, para cada métrica de software analisada, os percentuais de classes *com falhas* classificadas em cada faixa determinada pelos respectivos valores referência de Filó et al. [2015]. O complemento de cada célula, em relação a 100%, corresponde ao percentual de classes que não apresentaram falhas classificadas na respectiva faixa de valor. Além disso, foram calculados para cada uma das três faixas, a média e o desvio padrão, permitindo assim, analisar as faixas que foram melhores indicadores da ocorrência de falhas.

Os resultados observados para cada métrica nesta análise são compatíveis com aqueles observados quando se observou cada software individualmente onde 7 das 10 métricas avaliadas apresentaram o maior percentual de classes com falhas na faixa *Ruim*.

Considerando-se a faixa *Bom*, somente 2 das 10 (20%) métricas analisadas neste estudo, DIT e NSC, apresentaram o maior percentual de classes com falhas. Em média, esta faixa obteve aproximadamente 18% de classes com falha. Esse resultado é condizente com a definição do catálogo de Filó et al. [2015], no qual a faixa *Bom* corresponde aos valores que apresentaram alta frequência nos sistema. Apesar de esses valores não expressarem necessariamente as melhores práticas de Engenharia de Software, eles podem indicar um padrão de qualidade e, consequentemente, entidades

Tabela 8.3. Percentual de classes que apresentaram falhas por faixa e por métrica

Métrica	***Bom***	***Regular***	***Ruim***
	%	%	%
DIT	21,00	20,88	20,73
LCOM	16,39	23,61	35,08
NOF	17,09	29,59	41,53
NOM	15,22	24,82	39,77
NORM	20,10	28,79	31,21
NSC	21,17	20,69	18,17
NSF	17,52	30,34	39,31
NSM	19,26	33,11	39,18
SIX	19,45	25,10	21,43
WMC	11,57	24,73	44,37
Média	17,88	26,17	33,08
Desvio Padrão	2,98	4,13	9,65

assim classificadas tendem a não apresentar falhas. Dessa forma, conclui-se a aplicação da faixa *Bom* não é um indicador de falhas em versões futuras dos sistemas avaliados.

Em relação a faixa *Regular*, somente a métrica SIX (10%) apresentou o maior percentual de classes com falhas. Porém, em média, obteve-se aproximadamente 26% das classes com falhas, considerando-se todas as 10 métricas sob análise. Isso é um valor relativamente significativo, tendo em vista que é 8% maior do que o valor obtido para a faixa *Bom*. Esse aumento no percentual de classes com falhas identificadas era esperado, pois segundo Filó et al. [2015], a faixa *Regular* corresponde a valores que não são muito frequentes e nem são de baixa frequência. Dessa forma, as classes que têm medidas consideradas *Regular* são mais propensas a apresentar problemas em relação a faixa *Bom*. Conforme apresentado na Tabela 8.3, a aplicação da faixa *Regular* na indicação de falhas foi a segunda faixa com o maior percentual de ocorrência de falhas ainda que uma das métricas, SIX, indicou o maior percentual de falhas na faixa *Regular* em comparação com as demais faixas.

Por fim, em relação a faixa *Ruim*, 7 das 10 (70%) métricas analisadas, *LCOM, NOF, NOM, NORM, NSF, NSM e WMC*, obtiveram o maior percentual de classes com falhas. O percentual médio corresponde a, aproximadamente, 33% das classes. Esse valor é até 15% maior do que o obtido nas faixas *Bom* e *Regular*. Esse resultado está em consonância com o catálogo de valores referência de Filó et al. [2015], pois ele descreve a faixa *Ruim* como valores de baixa frequência, sendo a mais crítica em termos da propensão à ocorrência de problemas. Portanto, a aplicação da faixa *Ruim* mostra-se eficaz na indicação de ocorrência de falhas em versões futuras dos sistemas

avaliados.

Em geral, observa-se por meio deste estudo um crescimento no percentual de classes com falhas classificadas nas faixas *Bom*, *Regular* e *Ruim*, nessa ordem. Ou seja, de acordo com os resultados obtidos, quanto mais crítica a faixa de valores referência, maior a propensão à ocorrência de falhas nas classes dos sistemas. Os resultados obtidos sugerem que os valores referência propostos por Filó et al. [2015] são úteis à predição de falhas. Portanto, a resposta *QP2* é afirmativa. O catálogo de Filó et al. [2015] pode apoiar a predição de falhas em sistemas de software.

8.4 Ameaças à Validade do Experimento

A seguir, discutem-se os quatro tipos de ameaça à validade desse experimento, de acordo com as diretrizes de Wohlin et al. [2012]. São elas: ameaças de construção, ameaças internas, ameaças externas e ameaças de conclusão.

Ameaças de construção. Para este estudo, selecionou-se a ferramenta *BugMaps* de Couto et al. [2013b] para apoiar a coleta de falhas em sistemas disponíveis nos repositórios do *BugZilla e Jira*. Por se tratar de uma ferramenta automatizada, a *BugMaps* pode conter falhas. Ainda assim, ela foi escolhida com base em trabalhos anteriores que indicaram sua aplicabilidade e eficácia [Januário & Ferreira, 2016; Couto et al., 2013b].

Outra ameaça está relacionada à confiabilidade dos dois repositórios utilizados, que são abertos ao público para contribuição. Porém, ambos têm sido utilizados na literatura e em fábricas de software. Por fim, para classificar os valores das métricas de acordo com as faixas de valores *Bom*, *Regular* e *Ruim*, foi implementada nesta dissertação uma rotina em *Java* a fim de automatizar esse processo. Porém, como todo software, essa rotina pode conter falhas que impactem nos resultados obtidos. Como forma de tratamento dessa ameaça, a rotina foi testada em um pequeno conjunto de dados.

Ameaças internas. Durante a coleta de dados para análise, algumas ameaças relacionadas a fatores humanos podem ter afetado o processo. No cruzamento dos dados das falhas dos sistemas com a classificação das métricas de software por faixa, *Bom, Regular e Ruim*, foram removidas manualmente as classes com falhas que não tinham métricas de software computadas no *Qualitas.class Corpus 2013*. Isso foi necessário pois o *corpus* contém informações relativas à versão de 2013 dos sistemas, enquanto que os dados de falha abrangem informações das versões de 2013 a 2016.

Ameaças de conclusão. A análise provida nesse estudo é baseada no percentual de classes com falhas que foram classificadas em cada uma das faixas propostas por Filó et al. [2015] - ou seja, *Bom, Regular e Ruim*. Foram computadas métricas conhecidas na literatura, média e desvio padrão, a fim de apoiar a investigação da questão de pesquisa *QP2*. Ainda que existam outras formas de analisar esse tipo de estudo, considerou-se suficiente a análise por meio dessas métricas. Além disso, toda a análise foi computada por meio de planilhas. A fim de minimizar possíveis problemas, foi realizada uma inspeção manual dos arquivos gerados. Por fim, ressalta-se que nesse estudo foi considerada apenas a presença ou ausência de falhas na classe e não a quantidade de falhas identificadas por classe. Ainda que considerar a quantidade de falhas pudesse justificar a condução de um estudo complementar, considera-se que os resultados obtidos no presente estudo fornecem evidências suficientes para os objetivos deste trabalho.

Ameaças externas. Nesse estudo foram utilizados somente sistemas implementados em Java e catalogados pelo *Qualitas.class Corpus*. Além disso, desses sistemas, foram descartados aqueles cujos dados de falhas não estavam disponíveis nos repositórios *Bugzilla e Jira*, totalizando 10 sistemas analisados. Embora essa quantidade de sistemas tenha sido considerada suficiente para o estudo realizado, as conclusões obtidas por meio desse podem não ser generalizadas a qualquer contexto de desenvolvimento, por exemplo, sistemas implementados em outras linguagens de programação ou com características divergentes em relação aos sistemas analisados.

8.5 Lições Aprendidas

É possível nesse estudo observar a eficácia da utilização da faixa dos valores referência propostos por Filó et al. [2015] na predição de falhas. Observou-se para LCOM e WMC, que são métricas relacionadas a falta de coesão entre métodos de classes e à complexidade de classes, respectivamente, que quanto mais crítica é a faixa do valor referência, mais propensa à uma classe é a ocorrência de falhas. Além disso, observa-se que, para 7 das 10 métricas analisadas, a faixa *Ruim*, que é a mais crítica de acordo com Filó et al. [2015], mostrou-se mais eficaz na indicação de falhas em comparação as outras duas faixas menos críticas. Todavia a faixa *Regular* também se mostrou importante para a predição de falhas.

Os resultados obtidos sugerem aplicações práticas do uso de valores referência na predição de falhas, como por exemplo no apoio às atividades de garantia da qualidade de software. A identificação de classes cujas métricas estão classificadas na faixa *Ruim*

podem apoiar os desenvolvedores e testadores a concentrarem seus esforços em classes que tendem a apresentar falhas, minimizando assim, a ocorrência de problemas futuros nos sistemas.

Capítulo 9

Conclusão

Nesta dissertação avaliou-se a utilidade dos valores referência de métricas de software na identificação de *bad smells* em código-fonte e na predição de falhas em sistemas de software. Inicialmente foi realizado um mapeamento sistemático da literatura para se identificar o estado da arte. Esse mapeamento apontou que a definição de valores referência para métricas de software tem sido estudada recentemente na literatura, mas ainda não há uma avaliação aprofundada sobre os valores referência propostos.

O catálogo de valores referência definido por Filó et al. [2015] foi considerado neste trabalho por ser o catálogo com a maior quantidade de métricas e ter sido parcialmente avaliado anteriormente. Esse catálogo foi utilizado na definição de estratégias de detecção de cinco *bad smells: Large Class, Long Method, Data Class, Feature Envy e Refused Bequest.*

Para avaliar a utilidade dos valores referência propostos por Filó et al. [2015], foram conduzidos dois estudos distintos. O primeiro estudo investigou a eficácia dos valores referência na identificação dos cinco *bad smells.* O segundo estudo investigou a utilização de valores referência na predição de falhas de acordo com as faixas *Bom, Regular e Ruim.*

Com os resultados obtidos neste trabalho, conclui-se que as questões de pesquisa investigadas são respondidas como segue.

QP1. *Os valores referência de métricas de software orientados por objetos auxiliam a identificar bad smells?*

A resposta dessa questão é afirmativa. Os resultados obtidos no primeiro estudo mostram que os valores referência foram significativamente eficazes no apoio à detecção de *bad smells*, com percentuais em geral altos para *Recall* e moderados para Precisão e *F-measure.* Esse resultado positivo foi obtido tanto em relação a listas de

referência providas por ferramentas de detecção de *bad smells*, quanto em relação às listas geradas manualmente pelo especialista.

QP2. *Os valores referência de métricas de software orientados por objetos auxiliam a predizer falhas em um software?*

Observou-se no segundo estudo que os valores referência foram eficazes na predição de falhas para a maioria das métricas de software analisadas. A faixa *Ruim* apresentou os maiores percentuais de classes com falhas nos sistemas avaliados. Observou-se também que quanto maior a criticidade da faixa dos valores referência, maior o percentual de classes com falhas.

Os estudos realizados nesta dissertação apontam, então, que valores referência de métricas podem ser instrumentos úteis na avaliação da qualidade do software.

Como contribuições desta dissertação destacam-se:

- um mapeamento sistemático da literatura com foco em trabalhos sobre valores referência das métricas associados a *bad smells* e predição de falhas. Os resultados obtidos proporcionam à comunidade acadêmica uma visão do estado da arte nesse campo. Ao mesmo tempo revela a ausência de estudos com foco na utilização de valores referência para identificação de *bad smells* e predição de falhas.
- avaliação dos valores referência em softwares abertos com o objetivo de verificar se, de fato, é possível por meio deles realizar a detecção de *bad smells* em softwares orientados por objeto.
- avaliação dos valores referência em softwares abertos com o objetivo de verificar se, de fato, é possível por meio deles realizar a predição de falhas em sistemas de software orientados por objeto.
- ferramenta *FindSmells* para realizar detecção de *bad smells*.
- publicação de um artigo com os resultados parciais desta dissertação no XIV *Workshop* de Teses e Dissertações em Qualidade de Software (WTDQS), do Simpósio Brasileiro de Qualidade de Software de 2016 [Souza et al., 2016].

Como sugestões de trabalhos futuros, é importante que sejam realizadas avaliações adicionais do catálogo de Filó et al. [2015] referentes a outras questões relacionadas

a qualidade de software, como padrões de projeto, projeto arquitetural, dentre outros. É importante também que as análises realizadas por Filó et al. [2015] e neste trabalho sejam estendidas a sistemas implementados em outras linguagens de programação, como C#, *JavaScript e PHP*. A replicação dos estudos apresentados nesta dissertação para avaliação de outros catálogos de valores referência também é importante para se avaliar a aplicabilidade desses catálogos.

Referências Bibliográficas

Abaei, G.; Selamat, A. & Fujita, H. (2015). An empirical study based on semi-supervised hybrid self-organizing map for software fault prediction. *Knowledge-Based Systems*, 74:28--39.

Abreu, F. B. & Carapuça, R. (1994). Object-oriented software engineering: Measuring and controlling the development process. Em *Proceedings of the 4th international conference on software quality*, volume 186, pp. 1--8.

Alan, O. & Catal, C. (2011). Thresholds based outlier detection approach for mining class outliers: An empirical case study on software measurement datasets. *Expert Systems with Applications*, 38(4):3440--3445.

Alves, T. L.; Ypma, C. & Visser, J. (2010). Deriving metric thresholds from benchmark data. Em *Software Maintenance (ICSM), 2010 IEEE International Conference on*, pp. 1–10. ISSN 1063-6773.

Amber, S. W. (1998). *Análise e projeto orientado a objeto: seu guia para desenvolver sistemas robustos com tecnologia de objetos*. Rio de Janeiro: Infobook. Tradução Oswaldo Zanelli.

Aranha, E. & Borba, P. (2007). An estimation model for test execution effort. Em *First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*, pp. 107--116. IEEE.

Baggen, R.; Correia, J. P.; Schill, K. & Visser, J. (2012). Standardized code quality benchmarking for improving software maintainability. *Software Quality Journal*, 20(2):287--307.

Basili, V. R.; Briand, L. C. & Melo, W. L. (1996). A validation of object-oriented design metrics as quality indicators. *IEEE Transactions on software engineering*, 22(10):751--761.

Bellon, S.; Koschke, R.; Antoniol, G.; Krinke, J. & Merlo, E. (2007). Comparison and evaluation of clone detection tools. *IEEE Transactions on Software Engineering*, 33(9):577--591.

Benlarbi, S.; El Emam, K.; Goel, N. & Rai, S. (2000). Thresholds for object-oriented measures. Em *Software Reliability Engineering, 2000. ISSRE 2000. Proceedings. 11th International Symposium on*, pp. 24--38. IEEE.

Bertrán, I. M. (2009). *Avaliação da qualidade de software com base em modelos uml.* Tese de doutorado, PUC-Rio.

Biolchini, d. A.; Calmon, J.; Mian, P. G.; Natali, A. C. C.; Conte, T. U. & Travassos, G. H. (2007). Scientific research ontology to support systematic review in software engineering. *Advanced Engineering Informatics*, 21(2):133--151.

Cardoso, B. & Figueiredo, E. (2015). Co-occurrence of design patterns and bad smells in software systems: An exploratory study. Em *Proceedings of the annual conference on Brazilian symposium on information systems: Information systems: A computer socio-technical perspective*, pp. 347--354.

Catal, C.; Alan, O. & Balkan, K. (2011). Class noise detection based on software metrics and roc curves. *Information Sciences*, 181(21):4867--4877.

Catal, C. & Diri, B. (2009). A systematic review of software fault prediction studies. *Expert systems with applications*, 36(4):7346--7354.

Chhikara, A.; Chhillar, R. & Khatri, S. (2011). Evaluating the impact of different types of inheritance on the object oriented software metrics. *International Journal of Enterprise Computing and Business Systems*, 1(2):1--7.

Chidamber, S. R. & Kemerer, C. F. (1994). A metrics suite for object oriented design. *IEEE Transactions on software engineering*, 20(6):476--493.

Couto, C.; Maffort, C.; Garcia, R. & Valente, M. T. (2013a). Comets: A dataset for empirical research on software evolution using source code metrics and time series analysis. *ACM SIGSOFT Software Engineering Notes*, 38(1):1--3.

Couto, C.; Pires, P.; Valente, M. T.; Bigonha, R.; Hora, A. & Anquetil, N. (2013b). Bugmaps-granger: A tool for causality analysis between source code metrics and bugs. Em *Brazilian Conference on Software: Theory and Practice (CBSoft'13)*.

D'Ambros, M.; Lanza, M. & Robbes, R. (2010). An extensive comparison of bug prediction approaches. Em *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*, pp. 31--41. IEEE.

Fawcett, T. (2006). An introduction to roc analysis. *Pattern recognition letters*, 27(8):861--874.

Fernandes, E.; Oliveira, J.; Vale, G.; Paiva, T. & Figueiredo, E. (2016). A review-based comparative study of bad smell detection tools. Em *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering*, p. 18. ACM.

Ferreira, K. A. M. (2011). *Um modelo de predição de amplitude da propagação de modificações contratuais em software orientado por objetos.* Tese de doutorado, DCC/ UFMG.

Ferreira, K. A. M.; Bigonha, M. A. S.; Bigonha, R. S.; Mendes, L. F. O. & Almeida, H. C. (2012). Identifying thresholds for object-oriented software metrics. *Journal of Systems and Software*, 85(2):244--257. ISSN 0164-1212.

Filó, T. G. (2014). *Identificação de valores referência para métricas de softwares orientados por objetos.* Dissertação de mestrado, DCC/UFMG.

Filó, T.; da Silva Bigonha, M. & Ferreira, K. (2015). A catalogue of thresholds for object-oriented software metrics. Em *The First Int. Conf. on Advances and Trends in Software Engineering*, pp. 48--55.

Fontana, F. A.; Braione, P. & Zanoni, M. (2012). Automatic detection of bad smells in code: An experimental assessment. *Journal of Object Technology*, 11(2):5--1.

Fontana, F. A.; Ferme, V.; Zanoni, M. & Yamashita, A. (2015). Automatic metric thresholds derivation for code smell detection. Em *Proceedings of the Sixth International Workshop on Emerging Trends in Software Metrics*, pp. 44--53. IEEE Press.

Foucault, M.; Palyart, M.; Falleri, J.-R. & Blanc, X. (2014). Computing contextual metric thresholds. Em *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, pp. 1120--1125. ACM.

Fowler, M. (1999). *Refactoring: improving the design of existing code.* Pearson Education India.

Frakes, W. B. & Baeza-Yates, R. (1992). Information retrieval: data structures and algorithms.

Graves, T. L.; Karr, A. F.; Marron, J. S. & Siy, H. (2000). Predicting fault incidence using software change history. *IEEE Transactions on software engineering*, 26(7):653--661.

Gyimothy, T.; Ferenc, R. & Siket, I. (2005). Empirical validation of object-oriented metrics on open source software for fault prediction. *IEEE Transactions on Software engineering*, 31(10):897--910.

Herbold, S.; Grabowski, J. & Waack, S. (2011). Calculation and optimization of thresholds for sets of software metrics. *Empirical Software Engineering*, 16(6):812--841.

Januário, M. L. C. & Ferreira, K. A. M. (2016). Aprimoramento de uma ferramenta de medição de software. relatório técnico de iniciação científica. centro federal de educação tecnológica de minas gerais.

Kaur, S.; Singh, S. & Kaur, H. (2013). A quantitative investigation of software metrics threshold values at acceptable risk level. Em *International Journal of Engineering Research and Technology*, volume 2. ESRSA Publications.

Kayarvizhy, N. & Kanmani, S. (2011). Analysis of quality of object oriented systems using object oriented metrics. Em *Electronics Computer Technology (ICECT), 2011 3rd International Conference on*, volume 5, pp. 203--206. IEEE.

Kitchenham, B. (2004). Procedures for performing systematic reviews. *Keele, UK, Keele University*, 33(2004):1--26.

Kitchenham, B. & Charters, S. (2007). Guidelines for performing systematic literature reviews in software engineering. Em *In Software Engineering, Technical Report EBSE- 2007-01, Departament of Computer Science Keele University, Keele*.

Lanza, M. & Marinescu, R. (2007). *Object-oriented metrics in practice: using software metrics to characterize, evaluate, and improve the design of object-oriented systems*. Springer Science & Business Media.

Liu, H.; Ma, Z.; Shao, W. & Niu, Z. (2012). Schedule of bad smell detection and resolution: A new way to save effort. *IEEE Transactions on Software Engineering*, 38(1):220--235.

Lorenz, M. & Kidd, J. (1994). *Object-oriented software metrics: a practical guide*. Prentice-Hall, Inc.

Macia, I.; Garcia, J.; Popescu, D.; Garcia, A.; Medvidovic, N. & von Staa, A. (2012). Are automatically-detected code anomalies relevant to architectural modularity?: an exploratory analysis of evolving systems. Em *Proceedings of the 11th annual international conference on Aspect-oriented Software Development*, pp. 167--178. ACM.

Malhotra, R. & Bansal, A. J. (2015). Fault prediction considering threshold effects of object-oriented metrics. *Expert Systems*, 32(2):203--219.

Marinescu, R. (2004). Detection strategies: Metrics-based rules for detecting design flaws. Em *Software Maintenance, 2004. Proceedings. 20th IEEE International Conference on*, pp. 350--359. IEEE.

Martin, R. (1994). Oo design quality metrics. *An analysis of dependencies*, 12:151--170.

McCabe, T. J. (1976). A complexity measure. *IEEE Transactions on software Engineering*, (4):308--320.

Nagappan, M.; Zimmermann, T. & Bird, C. (2013). Diversity in software engineering research. Em *Proceedings of the 2013 9th joint meeting on foundations of software engineering*, pp. 466--476. ACM.

Nagappan, N.; Ball, T. & Zeller, A. (2006). Mining metrics to predict component failures. Em *Proceedings of the 28th international conference on Software engineering*, pp. 452--461. ACM.

Nunes, H. G. (2014). *Identificação de bad smells em software a partir de modelos UML*. Dissertação de mestrado, DCC/UFMG.

Oliveira, P.; Valente, M. T. & Lima, F. P. (2014). Extracting relative thresholds for source code metrics. Em *Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week-IEEE Conference on*, pp. 254--263. IEEE.

Padilha, J.; Figueiredo, E.; Sant?Anna, C. & Garcia, A. (2013). Detecting god methods with concern metrics: An exploratory study. *Proc. of the 7th LA-WASP, co-allocated with CBSoft*.

Paiva, T.; Damasceno, A.; Padilha, J.; Figueiredo, E. & Santanna, C. (2015). Experimental evaluation of code smell detection tools. Em *III Workshop on Software Visualization, Evolution, and Maintenance (VEM)*.

Pressman, R. S. (2006). *Engenharia de Software*. MacGraw Hill, Rio de Janeiro, 6 edição.

Radatz, J.; Geraci, A. & Katki, F. (1990). Ieee standard glossary of software engineering terminology. *IEEE Std*, 610121990(121990):3.

Riaz, M.; Mendes, E. & Tempero, E. (2009). A systematic review of software maintainability prediction and metrics. Em *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*, pp. 367--377. IEEE Computer Society.

Rodriguez, D.; Ruiz, R.; Riquelme, J. C. & Harrison, R. (2013). A study of subgroup discovery approaches for defect prediction. *Information and Software Technology*, 55(10):1810--1822.

Rosenberg, L.; Stapko, R. & Gallo, A. (1999). Risk-based object oriented testing. *24th SWE*.

Sahin, D.; Kessentini, M.; Bechikh, S. & Deb, K. (2014). Code-smell detection as a bilevel problem. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 24(1):6.

Sales, V.; Terra, R.; Miranda, L. F. & Valente, M. T. (2013). Recommending move method refactorings using dependency sets. Em *WCRE*, volume 20, p. 13.

Salfner, F.; Lenk, M. & Malek, M. (2010). A survey of online failure prediction methods. *ACM Computing Surveys (CSUR)*, 42(3):10.

Shatnawi, R. (2015). Deriving metrics thresholds using log transformation. *Journal of Software: Evolution and Process*, 27(2):95--113.

Shatnawi, R. & Althebyan, Q. (2013). An empirical study of the effect of power law distribution on the interpretation of oo metrics. *ISRN Software Engineering*, 2013.

Shatnawi, R.; Li, W.; Swain, J. & Newman, T. (2010). Finding software metrics threshold values using roc curves. *Journal of software maintenance and evolution: Research and practice*, 22(1):1--16.

Shin, Y. & Williams, L. (2011). Can traditional fault prediction models be used for vulnerability prediction? *Empirical Software Engineering*, 18(1):25--59.

Singh, S. & Kahlon, K. (2014). Object oriented software metrics threshold values at quantitative acceptable risk level. *CSI transactions on ICT*, 2(3):191--205.

Sommervile, I. (2011). *Engenharia de Software*. Pearson Education Brasil.

Sousa, B. L.; Souza, P. P.; Bigonha, M. A. S. & Ferreira, K. A. M. (2016). Findsmells, uma ferramenta de detecção de bad smells baseada em métricas de software. relatório técnico de pós-graduação. llp 001-2016, universidade federal de minas gerais.

Souza, P. P.; Ferreira, K. A. M. & Bigonha, M. A. S. (2016). A utilidade dos valores referência de métricas na avaliação da qualidade de softwares orientados por objeto. *15 Simpósio Brasileiro de Qualidade de Software - XIV Workshop de Teses e Dissertações em Qualidade de Software (WTDQS)*. ISSN .

Tempero, E.; Anslow, C.; Dietrich, J.; Han, T.; Li, J.; Lumpe, M.; Melton, H. & Noble, J. (2010). The qualitas corpus: A curated collection of java code for empirical studies. Em *2010 Asia Pacific Software Engineering Conference*, pp. 336--345. IEEE.

Terra, R.; Miranda, L. F.; Valente, M. T. & Bigonha, R. S. (2013). Qualitas. class corpus: A compiled version of the qualitas corpus. *ACM SIGSOFT Software Engineering Notes*, 38(5):1--4.

Tsantalis, N.; Chaikalis, T. & Chatzigeorgiou, A. (2008). Jdeodorant: Identification and removal of type-checking bad smells. Em *Software Maintenance and Reengineering, 2008. CSMR 2008. 12th European Conference on*, pp. 329--331. IEEE.

Tufano, M.; Palomba, F.; Bavota, G.; Oliveto, R.; Di Penta, M.; De Lucia, A. & Poshyvanyk, D. (2015). When and why your code starts to smell bad. Em *Proceedings of the 37th International Conference on Software Engineering-Volume 1*, pp. 403--414. IEEE Press.

Vale, G.; Albuquerque, D.; Figueiredo, E. & Garcia, A. (2015). Defining metric thresholds for software product lines: a comparative study. Em *Proceedings of the 19th International Conference on Software Product Line*, pp. 176--185. ACM.

Vale, G. A. (2016). *A benchmark-based method to derive metric thresholds*. Dissertação de mestrado, DCC/UFMG.

Vidal, S. A.; Marcos, C. & Díaz-Pace, J. A. (2014). An approach to prioritize code smells for refactoring. *Automated Software Engineering*, pp. 1--32.

Wikipédia (2016). Precisão e revocação. Acessado em 05 de Setembro de 2015.

Wohlin, C.; Runeson, P.; Höst, M.; Ohlsson, M. C.; Regnell, B. & Wesslén, A. (2012). *Experimentation in software engineering*. Springer Science & Business Media.

Zhang, F.; Mockus, A.; Zou, Y.; Khomh, F. & Hassan, A. E. (2013). How does context affect the distribution of software maintainability metrics? Em *ICSM*, pp. 350--359. Citeseer.

Zheng, J.; Williams, L.; Nagappan, N.; Snipes, W.; Hudepohl, J. P. & Vouk, M. A. (2006). On the value of static analysis for fault detection in software. *IEEE transactions on software engineering*, 32(4):240--253.