IDENTIFICAÇÃO DE VALORES REFERÊNCIA PARA MÉTRICAS DE SOFTWARES ORIENTADOS POR OBJETOS

TARCÍSIO GUERRA SAVINO FILÓ

IDENTIFICAÇÃO DE VALORES REFERÊNCIA PARA MÉTRICAS DE SOFTWARES ORIENTADOS POR OBJETOS

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais 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

Dezembro de 2014

Filó, Tarcísio Guerra Savino

F488i Identificação de valores referência para métricas de softwares orientados por objetos / Tarcísio Guerra Savino Filó. — Belo Horizonte, 2014
xxv, 197 f. : il. ; 29cm

Dissertação (mestrado) — Universidade Federal de Minas Geraiss — Departamento de Ciência da Computação.
Orientadora: Mariza Andrade da Silva Bigonha
Coorientadora: Kecia Aline Marques Ferreira

1. Computação — Teses. 2. Engenharia de software — Teses. 3. Software — Controle de qualidade — Teses. I. Orientadora. 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

Identificação de valores referência para métricas de softwares orientados por objetos

TARCÍSIO GUERRA SAVINO FILÓ

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 Ciência da Computação - CEFET-MG

PROF. PAULO CÉSAR MASIERO
Departamento de Sistemas de Computação - ICMC

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

Belo Horizonte, 15 de dezembro de 2014.

Resumo

Valores referência para a maioria das métricas de software ainda não são conhecidos. Essa pode ser a razão pela qual um processo de medição que deveria fazer parte de um processo de controle de qualidade de software ainda não se faz presente de forma disseminada na indústria de software. Neste trabalho, aplicamos um método empírico pré-definido para derivação de valores referência em um conjunto de 111 softwares e identificamos valores referência para 18 métricas de softwares orientados por objetos. Além disso, propomos algumas melhorias nesse método. Diferentemente de trabalhos anteriores, propusemos um catálogo de valores referência que agrupa uma quantidade maior de métricas, possibilitando a avaliação de métodos, classes e pacotes. Nossa abordagem sugere três faixas nos valores referência: Bom/Frequente, Regular/Ocasional e Ruim/Raro. A faixa Bom/Frequente corresponde aos valores que apresentam alta frequência. A faixa Ruim/Raro corresponde aos valores de baixa frequência, enquanto a faixa Regular/Ocasional corresponde a valores que não são muito frequentes e nem são de baixa frequência. Apesar de esses valores não expressarem necessariamente as melhores práticas da Engenharia de Software, eles refletem um padrão de qualidade seguido pela maioria dos software avaliados. Para avaliar a efetividade dos valores referência definidos em indicar o real panorama da qualidade de software, conduzimos três estudos de caso e um experimento. Os resultados obtidos sugerem que os valores referência propostos são capazes de indicar a qualidade dos softwares, provendo um referencial para a avaliação quantitativa da qualidade interna de softwares orientados por objeto, considerando métodos, classes e pacotes.

Palavras-chave: valores referência, métricas de software, medição de software, qualidade de software, orientação por objetos.

Abstract

Thresholds for the majority of software metrics are still not known. This might be the reason why a measurement method that should be part of a software quality assessment process is still not there in software industry. In this work, we applied a predefined empirical method to an 111 system dataset, identifying thresholds for 18 object-oriented software metrics. Furthermore, we proposed some improvements in that method. Differently from previous work, we have defined a catalogue of thresholds that gathers a greater amount of object-oriented software metrics, allowing the assessment of methods, classes and packages. Our approach suggests three ranges in the thresholds: Good/Common, Regular/Casual and Bad/Uncommon. The Good/Common range corresponds to values with high frequency. The Bad/Uncommon range corresponds to values with quite low frequency, and the Regular/Casual range is an intermediate one, which corresponds to values that are not too frequent neither have very low frequency. Although they do not necessarily express the best practices in Software Engineering, they reflect a quality standard followed by most of the evaluated software. To evaluate the effectiveness of the defined thresholds to indicate the real panorama of the software quality, we conducted three case studies and an experiment. The results suggest that the thresholds proposed can indicate the quality of the software, providing a benchmark for the quantitative assessment of the internal quality of object-oriented software, considering methods, classes and packages.

Keywords: thresholds, software metrics, software measurement, software quality, object-orientation.

Lista de Figuras

Lista de Tabelas

Lista de Siglas

A	Abstractness (Abstração)
AC	Afferent Coupling (Acoplamento Aferente)
ACI	Average Complexity of a Program due to Inheritance (Complexidade Média do Programa pela Herança)
AHF	Attribute Hiding Factor (Fator Ocultação de Atributo)
CALM	Class Aggregation Level Measure (Nível de Acoplamento da Classe)
CBO	Coupling between object classes (Acoplamento entre Classes de Objetos)
CCI	Class Complexity due to Inheritance (Complexidade da Classe pela Herança)
CCOM	Class Cohesion Measure (Coesão da Classe)
CICM	Class Inheritance Complexity Measure (Complexidade da Herança da Classe)
CMC	Class Method Complexity (Complexidade dos Métodos da Classe)
CMCM	Class Member Complexity Measure (Complexidade dos Membros da Classe)
COF	Coupling Factor (Fator Acoplamento)
CTA	Coupling Throught Abstract Data (Acoplamento por Dados Abstratos)
CTM	Coupling Throught Message Passing (Acoplamento por Troca de Mensagens)
DIT	Depth of Inheritance Tree (Profundidade de Árvore de Herança)
EC	Efferent Coupling (Acoplamento Eferente)
I	Instability (Instabilidade)
LCOM	Lack of Cohesion in Methods (Ausência de Coesão em Métodos)

MHF Method Hiding Factor (Fator Ocultação de Método)

MIF Method Inheritance Factor (Fator Herança de Método)

MLOC Lines of Code per Method (Linhas de Código por Método)

NAC Number of Ancestor Classes (Número de Classes Ancestrais)

NBD Nested Block Depth (Profundidade de Blocos Aninhados)

NDC Number of Descendent Classes (Número de Classes Descendentes)

NLM Number of Local Methods (Número de Métodos Locais)

NOC Number of Children (Número de Filhos)

PF Polymorphism Factor (Fator Polimorfismo)

RF Reuse Factor (Fator Reúso)

RFC Response For a Class (Resposta de Classe)

RMD Normalized Distance (Distância Normalizada)

SIX Specialization Index (Índice de Especialização)

WMC Weighted Methods Per Class (Métodos Ponderados por Classe)

Sumário

Capítulo 1

Introdução

Medição é parte fundamental de qualquer disciplina de engenharia, e disciplinas da Engenharia de Software não são exceções. Nesse contexto, métricas de software referem-se a medições que podem ser aplicadas para verificar os indicadores de processos, projetos e produtos de software. Pressman [2006] define três termos comumente utilizados na literatura sobre métricas de software:

- métrica: é um padrão de medição para avaliar um atributo de algo relacionado a software.
- medida: é o resultado da medição. Uma medida indica quantitativamente a presença de um atributo em determinado software.
- medição: é o ato ou efeito de medir algo de acordo com uma métrica.

Segundo Pressman [2006], métricas proporcionam medidas quantitativas para avaliar a qualidade dos projetos, permitindo que engenheiros de software realizem alterações no decorrer do processo de desenvolvimento que aumentarão a qualidade do produto final. Os principais objetivos das métricas de softwares 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.

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 confiáveis de manutenabilidade, confiabilidade e portabilidade [Sommerville, 2010].

Softwares orientados por objetos são fundamentalmente diferentes de softwares desenvolvidos segundo métodos tradicionais. Por isso, as métricas de softwares ori-

entados por objetos devem avaliar recursos que os distinguem dos softwares convencionais. Esses recursos - classes, herança, encapsulamento, polimorfismo, abstração, entre outros - permitem atingir fatores de qualidade de software com maior facilidade porque viabilizam a obtenção de sistemas de estrutura muito flexível. Tal flexibidade estrutural impacta em maior extensibilidade, maior reusabilidade e maior facilidade de manutenção.

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]. Uma possível razão é que, para a grande maioria das métricas, não são conhecidos valores referência. Há um grande campo de pesquisa em aberto na definição de métodos para derivação de valores referência para métricas de software orientado por objetos [Rosenberg et al., 1999; Benlarbi et al., 2000; Shatnawi et al., 2010; Chhikara et al., 2011; Ferreira et al., 2011, 2012; Kaur et al., 2013; Oliveira et al., 2014]. A definição de métodos para identificação de valores referência de métricas de software tem sido o foco principal dos trabalhos realizados nessa área até então entretanto somente identificaram-se valores referência para um número reduzido de métricas. Dessa forma, o que se observa no estado atual da arte é que há uma quantidade razoável de métodos definidos e dezenas de métricas propostas na literatura, porém, há poucas métricas com valores referência conhecidos, dificultando e até mesmo impossibilitando o gerenciamento da qualidade de software por meios quantitativos.

1.1 Objetivos

O objetivo principal desta dissertação de mestrado é definir um catálogo de valores referência para 18 métricas de softwares orientados por objetos. Esse é um número de métricas e valores referências expressivo se comparado com os números relatados na literatura, oriundos de trabalhos anteriores. Os valores referência propostos almejam prover um *benchmark* para a avaliação quantitativa da qualidade interna de softwares orientados por objetos, considerando não somente classes, mas também métodos e pacotes. Não é objetivo deste trabalho a proposição de um novo método, mas, sim, identificar valores referência para mais métricas. Portanto, para produzir este catálogo utilizou-se um método empírico, proposto por Ferreira et al. [2012], que se baseia na análise das distribuições estatísticas de medidas encontradas na prática. Para atingir o objetivo principal são definidos os objetivos específicos descritos a seguir.

1. Estender a metodologia para extração de valores referência proposta por Ferreira

et al. [2012]. O objetivo dessa extensão é criar um conjunto de procedimentos sistemáticos para a metodologia proposta por Ferreira et al., avançando nas definições e técnicas estatísticas utilizadas.

2. Propor um catálogo de valores referência para métricas de softwares orientados por objetos. Há dezenas de métricas de software propostas na literatura [Chidamber & Kemerer, 1994; Brito e Abreu & Carapuca, 1994; Lamrani et al., 2011, 2013; Li, 1998; Mishra, 2011; Malik & Chhillar, 2011], mas poucas tiveram valores referência indicados. Neste trabalho, são definidos valores referência para 18 métricas de software orientado por objetos, disponibilizados em um catálogo.

3. Avaliar a eficácia dos valores referência propostos. A partir do conhecimento qualitativo dos softwares analisados, avaliar a eficácia dos valores referência extraídos na identificação de métodos, classes e pacotes em risco, bem como na identificação de métodos, classes e pacotes bem projetados. Para essa avaliação serão conduzidos três estudos de caso e um experimento. Espera-se que os valores referência sejam capazes de indicar a qualidade interna de softwares orientados por objetos por meios quantitativos.

1.2 Importância da Definição de Valores Referência

Os valores referência para diversas métricas propostas na literatura ainda não são conhecidos ou utilizam metodologias distintas que derivam diferentes valores. Isso é um problema e, como consequência, não há condições adequadas para diferenciar, quantitativamente, o que é um bom software e o que é um software ruim. Conhecer esses valores é essencial para que as métricas de software possam ser utilizadas para resolver esse problema. Além disso, sem o conhecimento desses valores referência, não é possível responder perguntas simples como "Quais classes do sistema possuem muitos métodos?" ou "Quais métodos do sistema possuem muitos parâmetros?".

Com a devida utilização de valores referência catalogados para métricas de software orientado por objetos, possíveis problemas no software poderiam 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 direcionar esforços de testes e manutenções preventivas para pontos críticos do sistema, otimizando a aplicação dos recursos disponíveis durante a fase de operação. Outro cenário possível é a utilização dos valores referência como critérios para avaliar a qualidade interna do software, por exemplo, em licitações, permitindo a tomada de decisão, por exemplo em processos de aquisição de softwares.

No panorama atual da Engenharia de Software, aspectos da qualidade interna do software em geral são avaliados por meio de inspeções qualitativas no software, o que demanda tempo e gera custos. Com a aplicação das métricas em conjunto com um catálogo de valores referência definido empiricamente, cria-se uma alternativa em que avaliações podem ser realizadas de forma automatizada, baseada em aspectos quantitativos do software. Vislumbra-se que as métricas possam ser efetivamente aplicadas na indústria de software.

1.3 Contribuições

As principais contribuições oriundas da pesquisa realizada são:

1. um catálogo de valores referência para 18 métricas de softwares orientados por objetos, derivados a partir de um *dataset* de 111 softwares *open-source*;
2. proposta de um catálogo que apresenta as distribuições estatísticas seguidas pelos conjuntos de medidas de cada uma das 18 métricas de softwares orientados por objetos, permitindo identificar características acerca de como os softwares estão sendo construídos em relação aos aspectos avaliados pela métricas;
3. melhorias no método de derivação de valores referência proposto por Ferreira et al. [2012], avançando nas definições e técnicas utilizadas;
4. desenvolvimento de uma ferramenta de filtragem de métodos, classes e pacotes em risco, baseada nos valores referência sugeridos;
5. resultados da avaliação dos valores referência propostos em softwares abertos e proprietários, exemplificando o uso e atestando a eficiência da aplicação desses valores.

1.4 Organização da Dissertação

Esta dissertação de mestrado está estruturada da seguinte forma:

Capítulo 2 apresenta uma revisão de literatura sobre métricas de software orientado por objetos.

Capítulo 3 apresenta o estado da arte em valores referência para métricas de softwares orientado por objetos.

Capítulo 4 apresenta a solução proposta para o problema estudado neste trabalho, descrevendo o método para derivar os valores referência, os dados utilizados para análise e como os valores derivados serão avaliados.

Capítulo 5 apresenta o processo de derivação de cada uma das métricas estudadas, os resultados obtidos e uma análise qualitativa do que os valores derivados representam em termos de qualidade de software, fazendo uma conexão entre os valores referência e os bons princípios de projeto. Nesse capítulo também são discutidas as limitações e vantagens do método utilizado em relação ao que tem sido proposto em outros métodos.

Capítulo 6 apresenta a ferramenta *RAFTool* (*Risk Artifacts Filter*), que realiza a filtragem de métodos, classes e pacotes com base nos valores referência propostos.

Capítulo 7 apresenta a condução de um experimento que tem por objetivo avaliar os valores referência propostos como indicativo de qualidade em processos de reestruturação de pacotes em softwares orientados por objetos.

Capítulo 8 apresenta a condução de um estudo de caso que avalia um software proprietário com problemas de qualidade interna, com o propósito de realizar a verificação da capacidade dos valores referência propostos em indicar esse panorama.

Capítulo 9 apresenta a condução de dois estudos de casos. O primeiro deles avalia a capacidade dos valores referência na indicação de métodos e classes de baixa qualidade. O segundo deles visa a avaliação de um software de alta qualidade, com o propósito de verificação da capacidade dos valores referência em indicar esse panorama. Ambos os estudos de casos são conduzidos em softwares do *dataset* utilizado no processo de derivação dos valores referência.

Capítulo 10 apresenta as conclusões deste trabalho e indica trabalhos futuros.

Capítulo 11 lista as publicações oriundas deste trabalho de dissertação.

Capítulo 2

Métricas de Software Orientado por Objetos

Um elemento fundamental para qualquer processo que tenha por meta atingir qualidade em seu produto final é a medição. Medidas são utilizadas com o objetivo de permitir 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. Da mesma forma, para garantir a qualidade de software é necessário que existam medidas com as quais seja possível avaliar aspectos do sistema, possibilitando controlar a qualidade do produto durante o curso dos projetos de software. Nesse contexto, as métricas de software se apresentam como elemento fundamental para avaliação quantitativa da presença de determinado atributo relacionado ao software.

Existem diversas métricas para orientação por objetos disponíveis na literatura, dentre as quais destacam-se o conjunto de métricas propostas por Chidamber & Kemerer [1994] e Brito e Abreu & Carapuca [1994]. Esses conjuntos são popularmente conhecidos como Métricas CK e Métricas MOOD e merecem destaque pela grande relevância em outras pesquisas. A Tabela 2.1 ilustra a relevância e colaboração científica do trabalho de Chidamber & Kemerer [1994], amplamente citado em outros trabalhos científicos. O trabalho de Brito e Abreu & Carapuca [1994], apesar de ser menos referenciado na literatura, apresenta métricas complementares ao conjunto CK e, juntamente com o trabalho de Chidamber & Kemerer [1994], foram precursores na área de métricas de softwares orientados por objetos. As métricas CK e MOOD serão tratadas nas Seções 2.1 e 2.2, respectivamente.

O objetivo deste capítulo é apresentar e discutir as principais métricas de softwares orientados por objetos propostas na literatura, dentre as quais aquelas que serão objeto de estudo desta pesquisa. As Seções 2.3 e 2.4 apresentam as métricas de Martin

Tabela 2.1: Número de citações de CK e MOOD

Repositório	CK	MOOD
IEEEXplore	637	-
ACM DL	702	-
Google Scholar	-	150

e algumas métricas que são indicadores de complexidade, respectivamente. Trabalhos mais recentes como os de Li [1998], Mishra [2011] e Malik & Chhillar [2011], que definem métricas de softwares orientados por objetos, serão discutidos nas Seções 2.5, 2.6 e 2.7, respectivamente. Algumas métricas de rede propostas por Newman [2003] são discutidas na Seção 2.8, bem como suas aplicabilidades no contexto de softwares orientados por objetos. As Seções 2.9 e 2.10 concluem o capítulo, trazendo um resumo das métricas discutidas durante o capítulo e uma análise crítica delas, respectivamente.

2.1 Métricas CK

Chidamber & Kemerer [1994] definem seis métricas de software orientado por objetos, conhecidas como conjunto CK. O conjunto CK contém as seguintes métricas:

- *Métodos Ponderados por Classe* (WMC): esta métrica consiste do somatório das complexidades dos métodos que constituem uma classe. Os autores deixam em aberto a definição de complexidade. Segundo Chidamber & Kemerer [1994], essa métrica é um indicador do esforço de desenvolvimento e manutenção da classe analisada. Quanto maior for o número de métodos na classe, maior a tendência da classe ficar menos específica, limitando assim o potencial de reúso e prejudicando o aspecto da coesão.

- *Profundidade de Árvore de Herança* (DIT): esta métrica representa a distância de uma determinada classe até a classe raiz na hierarquia de herança de um software. Segundo Chidamber & Kemerer [1994], quanto mais profunda uma classe estiver em uma hierarquia de classes, mais métodos serão herdados e, consequentemente, a classe se tornará mais complexa. Além disso, grandes distâncias entre classes filhas e a classe raíz na hierarquia de herança caracterizaM um projeto mais complexo e, consequentemente, mais propenso a erros.

- *Número de Filhos* (NSC): esta métrica representa o número de sub-classes subordinadas a uma determinada classe. Chidamber & Kemerer [1994] argumentam

que quanto maior o número de sub-classes, maior o reúso proporcionado pela herança. Além disso, o número de sub-classes é um bom indicativo da importância da classe no projeto. Portanto, quanto maior esse número, mais atenção a classe deve receber em termos de testes. Quando há um grande número de sub-classes correspondentes a uma determinada classe, pode haver abstrações impróprias ou uma utilização errônea dos conceitos de herança. Se NOC começa a aumentar indefinidamente, pode ser um indício de que algumas das sub-classes podem compartilhar atributos e comportamentos comuns, que deveriam estar em uma nova classe mãe dentro da hierarquia de herança.

- *Acoplamento entre Classes de Objetos* (CBO): esta métrica mede a quantidade de classes com que uma classe se relaciona, o que caracteriza acoplamento. Essa relação pode ocorrer por meio do acesso da classe em relação a uma variável ou método definidos em outra classe. Esta métrica é de fundamental importância para mensurar o nível de acoplamento de uma classe. Como destacaram Chidamber & Kemerer [1994], quanto mais acoplada uma determinada classe estiver a outras classes, menor será seu potencial de reúso. Além disso, para promover o encapsulamento e a modularidade, as interações entre os objetos devem ser minimizadas o tanto quanto possível. Classes com alto grau de acoplamento são mais sensíveis a mudanças em outros partes do sistema, tornando a manutenção do software mais difícil.

- *Resposta de Classe* (RFC): esta métrica mede o número de métodos que podem ser executados por uma instância de uma classe em resposta a uma mensagem recebida. Esse número é dado pelo conjunto de métodos de outras instâncias acionados por cada um dos métodos da classe, acrescido do conjunto de métodos da própria classe. Segundo Chidamber & Kemerer [1994], essa métrica é um bom indicativo da complexidade da classe. Quanto maior for o RFC da classe, mais complexa e, consequentemente, mais difícil serão os seus testes e a sua manutenção. Essa métrica analisa em um nível mais profundo a questão do acoplamento da classe.

- *Ausência de Coesão em Métodos* (LCOM): esta métrica mede a quantidade de pares de métodos de uma determinada classe em que a similaridade é zero, subtraída da quantidade de pares de métodos da classe em que a similaridade é diferente de zero. A similaridade entre dois métodos ocorre pelo uso comum de variáveis de uma instância da classe. Ela é diferente de nula quando eles acessam uma ou mais variáveis de instância comuns. A métrica visa ser um indicador da

ausência de coesão entre os métodos de uma classe. Dessa forma, quanto maior LCOM, menor será a coesão entre os pares de métodos, haja visto que a ausência de similaridade entre os pares contribui para aumentar o valor medido, enquanto a presença de pares de métodos com similaridade contribuem para diminuir esse valor. Segundo Chidamber & Kemerer [1994], um valor alto para LCOM é um indicativo de que a classe não é coesa, ou seja, não é específica a um determinado propósito, indicando que deve-se analisar a possibilidade de dividi-la em uma ou mais classes que atendam a propósitos específicos.

2.2 Métricas MOOD

Brito e Abreu & Carapuca [1994] definem oito métricas de software orientado por objetos, que avaliam aspectos de herança, encapsulamento, coesão, acoplamento, polimorfismo e reúso de software. Esse conjunto de métricas é conhecido como métricas MOOD. Uma das principais características desse conjunto de métricas é que elas são definidas por meio de uma razão, onde o numerador representa a contagem do aspecto avaliado no software e o denominador representa o valor máximo daquele aspecto. Isso faz com que os valores das métricas tenham independência em relação ao tamanho do software, pois os valores variam entre 0 e 1, trazendo a proporção da característica avaliada dentro do software. O conjunto MOOD é composto das seguintes métricas:

- *Fator Ocultação de Método* (MHF): esta métrica define a razão entre a quantidade de métodos ocultos em todas as classes do sistema e a quantidade total de métodos definidos em todas as classes do sistema. De acordo com Brito e Abreu & Carapuca [1994], ela proporciona uma visão do nível de encapsulamento aplicado no software, isto é, o quanto os detalhes de implementação estão escondidos, com a classe disponibilizando seus serviços por meio de uma interface limitada a suas responsabilidades. Quanto maior o número de métodos ocultos, maior será a aproximação da métrica do valor 1. Quanto mais próximo de 0 o valor da métrica, mais métodos estão visíveis para os usuários das classes do sistema, o que é um indicativo de baixo grau de encapsulamento do sistema.

- *Fator Ocultação de Atributo* (AHF): esta métrica define a razão entre a quantidade de atributos ocultos em todas as classes do sistema e a quantidade total de atributos definidos em todas as classes do sistema. Tal como a métrica MHF, ela proporciona uma visão a nível de sistema do encapsulamento das implementações. Segundo Ferreira [2006], a ocultação de atributos é fundamental para

garantir a independência das classes em um software orientado por objetos. Se duas ou mais classes tiverem acesso a manipulação direta de um mesmo atributo, é formado uma grande dependência entre as classes que o utilizam, o que pode tornar a manutenção do software mais difícil. Dessa forma, o ideal é que não haja atributos públicos nas classes, sendo o valor ideal para a métrica 1. Valores próximos de zero indicam baixa qualidade do software, apresentando baixo grau de encapsulamento e alto grau de acoplamento.

- *Fator Herança de Método* (MIF): esta métrica define a razão entre o somatório do número de métodos herdados de cada classe no sistema e o somatório do número de métodos disponível em cada classe no sistema. Essa métrica é um indicador do uso do recurso da herança nos softwares orientados por objeto. Conforme Brito e Abreu & Carapuca [1994], valores próximos de 0 indicam que a herança está sendo pouco utilizada. Isso minimiza o reúso e a abstração provida pela herança em softwares orientados por objetos.

- *Fator Herança de Atributo* (AIF): esta métrica define a razão entre o somatório do número de atributos herdados e o somatório do número de atributos disponível em cada classe no sistema. Tal como a métrica MIF, valores próximos de 0 indicam que a herança está sendo pouco utilizada e valores próximos de 1 indicam alta utilização do recurso.

- *Fator Acoplamento* (COF): esta métrica define a razão entre o número de relações cliente-servidor entre todas as classes do sistema e o número máximo possível de relações cliente-servidor entre as classes do sistema. A relação cliente-servidor entre duas classes acontece quando uma determinada classe acessa atributos ou métodos de outra classe. Essa relação caracteriza acoplamento entre as classes. Portanto, COF permite avaliar o quão acopladas estão as classes do sistema. Valores próximos de 0 indicam baixo nível de acoplamento, o que caracteriza uma estrutura modularizada, enquanto valores próximos de 1 indicam alto nível de acoplamento, o que poder levar ao aumento do esforço de manutenção e desenvolvimento, pois, nessa situação, os módulos são muito dependentes entre si.

- *Fator Polimorfismo* (PF): esta métrica define a razão entre o número de situações de polimorfismo encontrados no sistema e o número máximo de possíveis situações de polimorfismo que podem existir no sistema. Valores próximos de 1 indicam alta utilização de polimorfismo e valores próximos de 0 indicam baixa utilização desse recurso. Segundo Brito e Abreu & Carapuca [1994], o polimorfismo é de grande

importância para softwares orientado por objetos, pois possibilita a execução de comportamentos especializados em tempo de execução. Ou seja, dadas subclasses de uma determinada classe mãe, o mesmo método acionado pode tomar diversas formas de execução, dada a instância que recebe a mensagem em tempo de execução, sem a necessidade de tratá-los de forma diferenciada.

- *Fator Reúso* (RF): esta métrica é o somatório de duas razões, que caracterizam o reúso na forma de bibliotecas de classes (*lib*) e o reúso na forma de herança. A primeira razão é dada pelo número de classes utilizadas no software que fazem parte da *lib* e o número total de classes no sistema. A segunda razão é dada pelo número de classes que não estão na *lib*, multiplicada pela métrica MIF, e o número total de classes no sistema. O objetivo dessa segunda razão é avaliar o grau de reúso das classes por meio da herança. Segundo Brito e Abreu & Carapuca [1994], valores próximos de 1 indicam alto grau de reúso, o que é positivo, uma vez que indica que foram aproveitados módulos já desenvolvidos para construir software, minimizando o tempo e esforço de desenvolvimento e manutenção. Valores próximos de 0 indicam baixo grau de reúso, o que indica tempo e esforço de desenvolvimento e manutenção, haja visto que não se reaproveitam módulos já desenvolvidos e testados.

2.3 Métricas de Martin

Martin [1994] propõe um conjunto de métricas para medir a qualidade de softwares orientados por objetos em termos de acoplamento. Projetos com alto grau de acoplamento tendem a ser rígidos, não reutilizáveis e tem baixa qualidade em relação a manutenabilidade. O conjunto é composto pelas seguintes métricas:

- *Acoplamento Aferente* (AC): classes podem ser agrupadas em categorias, formando grupos coesos em que: se uma classe da categoria é alterada, as outras classes tendem a ser alteradas; as classes são reutilizadas em conjunto; as classes tem objetivos comuns. Nesse contexto, AC mede o número de classes externas a uma determinada categoria[1] que dependem das classes internas dessa categoria. Quanto maior o AC, maior a quantidade de classes às quais a categoria provê serviços, indicando sua relevância dentro do projeto. Com isso, ela deve receber testes mais rigorosos. Martin [1994] afirma que valores altos de AC indicam um projeto com alto grau de acoplamento.

[1] Essas categorias são chamadas *packages* (pacotes de classes) em Java.

- *Acoplamento Eferente* (EC): esta métrica mede o número de classes internas a uma determinada categoria que dependem de classes externas a essa categoria. Em outras palavras, esta métrica é um indicativo do nível de acoplamento da categoria a outras classes do projeto. Segundo Martin [1994], um valor alto de EC indica que a categoria consome mais serviços providos por classes externas a categoria, correspondendo a um alto grau de acoplamento.

- *Instabilidade* ($I = EC \div (AC + EC)$): esta métrica é uma razão-proporção entre o número de acoplamentos eferentes e a soma dos acoplamentos aferentes e eferentes da categoria avaliada. Nesta métrica, $I = 0$ indica o máximo de estabilidade e $I = 1$ indica o máximo de instabilidade de uma categoria. Em outras palavras, quanto menor for o acoplamento aferente (prestação de serviços) em relação ao acoplamento eferente (consumo de serviços), mais próximo de de 1 será o valor medido, e o pacote apresentará maior instabilidade. Isso significa que a categoria provê poucos serviços (AC com valores pequenos) em relação aos serviços por ela consumidos (EC).

- *Abstração* (A): esta métrica é uma razão-proporção entre o número de classes abstratas da categoria e o número total de classes da categoria. Categorias que apresentam abstração devem possuir classes externas que a estendem (acoplamento aferente). Conforme observado no item anterior, a instabilidade é minimizada com a presença de acoplamentos aferentes, o que sugere uma relação entre a minimização da instabilidade por meio do aumento da abstração. Contudo, Martin defende que se deve evitar a dependência a módulos instáveis. Por isso, categorias instáveis não devem ser abstratas, devem ser concretas. A relação ideal entre instabilidade e abstração deve ser da seguinte forma: categorias com grau mínimo de instabilidade devem apresentar grau máximo de abstração ($(I, A) = (0, 1)$); categorias com grau máximo de instabilidade devem apresentar grau mínimo de abstração ($(I, A) = (1, 0)$). O objetivo disso é desencorajar a dependência a categorias instáveis.

- *Distância Normalizada* ($RMD = |A + I + 1|$): dados os conceitos das métricas expostos anteriormente: instabilidade e abstração, Martin [1994] propõe uma noção de balanceamento entre os valores dessas métricas. Uma categoria pode ser parcialmente extensível porque ela é parcialmente abstrata. Sendo a categoria parcialmente estável, suas extensões não estão sujeitas a instabilidade máxima. Pela relação ideal estabelecida entre instabilidade e abstração, categorias com grau mínimo de instabilidade devem apresentar grau máximo de abstração

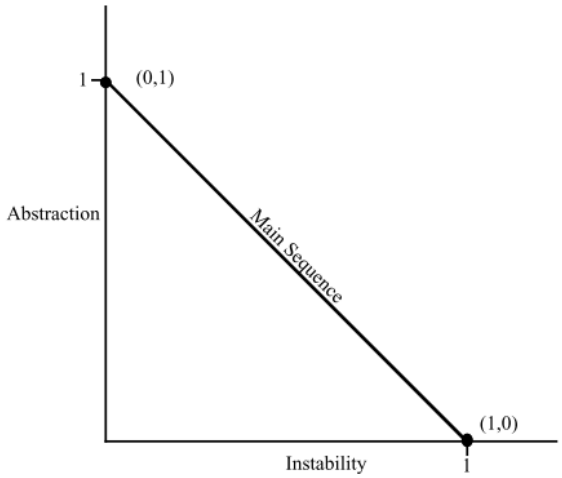


Figura 2.1: *Main Sequence*. Fonte: Martin [1994]

$((I, A) = (0, 1))$ e categorias com grau máximo de instabilidade devem apresentar grau mínimo de abstração $((I, A) = (1, 0))$. Uma linha entre esses pontos pode ser traçada em um gráfico, como pode ser visto na Figura 2.1. Essa linha, denominada *Main Sequence*, representa o balanceamento entre abstração e instabilidade considerado como aceitável por Martin. Uma categoria que fica na *Main Sequence* não é tão abstrata para sua estabilidade, nem é tão instável para sua abstração. Ela apresenta um grau de balanceamento aceitável de classes concretas e abstratas em relação a seus acoplamentos aferentes e eferentes. Martin lembra que as situações ideais são os extremos da *Main Sequence*, mas isso é muito difícil de se obter na prática. Pelos motivos expostos, a métrica Distância Normalizada mede a distância perpendicular de (I, A) da *Main Sequence*, ou seja, o quão longe a relação está do ponto mais próximo do grau de balanceamento considerado entre instabilidade e abstração considerado como aceitável. De acordo com Martin, quanto maior for essa distância, menor o grau de balanceamento entre Instabilidade a Abstração.

2.4 Métricas de Complexidade

Nesta seção são apresentadas algumas métricas comumente utilizadas na litetura como indicadores de complexidade nos softwares:

- *Linhas de Código por Método* (MLOC): esta métrica conta o número de linhas de código de cada um dos métodos do sistema.

- *Complexidade de McCabe [McCabe, 1976]*: esta métrica visa avaliar a complexidade de um programa. Para isso, o fluxo de execução do código é representado por meio de um grafo, no qual os nodos representam comandos do programa e uma aresta direcionada de um nodo `A` para um nodo `B` representa que o fluxo do programa pode ir de `A` para `B`. Esta métrica conta o número de caminhos independentes no grafo de execução do programa. A métrica é dada por $McCabe = E - N + P$, onde E é o número arestas no grafo, N é o número de nodos no grafo e P é a quantidade de componentes conectados no grafo. A complexidade de *McCabe* tem sido utilizada para avaliar a complexidade de métodos em softwares orientados por objetos.

- *Índice de Especialização* (SIX) [Lorenz & Kidd, 1994]: esta métrica é definida 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 ($SIX = NORM * DIT/NOM$). Esta métrica visa avaliar o quanto determinada classe sobrescreve o comportamento de suas superclasses. De acordo com os autores da métrica, quanto maior o valor do índice de especialização, maior é a execução de comportamentos especializados em tempo de execução, indicando uma classe mais complexa, devendo receber maior esforço de testes.

- *Profundidade de Blocos Aninhados* (NBD): esta métrica mede a profundidade máxima de blocos aninhados no programa. Por exemplo, três comandos *for* aninhados em um método caracterizam um profundidade máxima de blocos $NBD = 3$. Esta métrica é um indicativo da complexidade do programa, uma vez que conforme a quantidade de blocos aninhados aumenta, o trecho torna-se mais difícil de entender.

2.5 Métricas de LI

No trabalho de Li [1998], são propostas seis métricas para avaliar softwares orientados por objetos, que apresentam em suas definições correções para algumas deficiências apontadas pelo autor no trabalho de Chidamber & Kemerer [1994]. Por essa característica, as métricas propostas por Li são intimamente relacionadas com o conjunto CK. O conjunto é composto pelas seguintes métricas:

- *Número de Classes Ancestrais* (NAC): esta métrica corresponde à métrica DIT de Chidamber & Kemerer [1994]. DIT conta os níveis da árvore de herança até o nó raíz. Segundo o autor, DIT tem por objetivo capturar o número de classes

que tem influência sobre a classe analisada, dada a hierarquia de herança. No entanto, DIT não distingue se, em determinado nível, a classe herda de uma ou mais classes. Por isso, o autor propõe a métrica NAC, que mede o número de classes que antecedem a classe analisada na hierarquia de heranças, ou seja, não somente a quantidade de níveis.

- *Número de Classes Descendentes* (NDC): esta métrica corresponde à métrica NOC de Chidamber & Kemerer [1994]. NOC mede o número de sub-classes de uma determinada classe. Segundo o autor, NOC tem por objetivo capturar o número de classes que podem sofrer influência de uma alteração na classe analisada. No entanto, NOC considera somente as classes filhas imediatas, ignorando o restante das classes que herdam da classe analisada de forma não imediata. Por isso, o autor propõe a métrica NDC, que mede o número de classes que herdam da classe analisada em todos os níveis da hierarquia de herança, e não somente as classes filhas imediatas.

- *Número de Métodos Locais* (NLM): esta métrica define a quantidade de métodos de uma classe acessíveis a outras classes. Ou seja, conta a quantidade de métodos públicos disponibilizados pela classe. Segundo Li, valores altos de NLM podem indicar que a classe está realizando muitos serviços, ou seja, apresentando baixa coesão. Além disso, valores altos de NLM podem indicar um software com alto grau de acoplamento, o que implica em custo de desenvolvimento e manutenção.

- *Complexidade dos Métodos da Classe* (CMC): esta métrica sumariza a complexidade interna de todos os métodos da classe, independentemente do método ser público ou privado. Essa definição é a mesma da métrica WMC de Chidamber & Kemerer [1994], que é a soma da complexidade de todos os métodos na classe. No entanto, a definição difere pois para CMC é definido que a complexidade de cada método deve ser sumarizada por meio de outra métrica, como por exemplo, LOC (*Line of Codes*). Para Li, CMC proporciona uma melhor noção do custo de desenvolvimento e manutenção da classe do que a métrica WMC, pelo conceito de complexidade ser definido por meio de outra métrica.

- *Acoplamento por Dados Abstratos* (CTA): para definição desta métrica, é considerado o seguinte tipo de acoplamento: *Coupling Througth Abstract Data*. Nesse tipo de acoplamento, uma classe `B` está acoplada a uma classe `A` quando a classe `B` utiliza a classe `A` na definição de seus atributos. A métrica conta o número total de classes utilizadas na definição dos atributos de uma determinada classe.

Isso proporciona uma visão do nível de acoplamento dessa classe em relação a definição de atributos.

- *Acoplamento por Troca de Mensagens* (CTM): para a definição desta métrica é considerado o seguinte tipo de acoplamento: *Coupling Throught Message Passing*. Nesse tipo de acoplamento, uma classe `B` está acoplada a uma classe `A` quando a classe `B` invoca um ou mais métodos de `A`. A métrica conta o número total de mensagens enviadas a outras classes a partir de uma determinada classe, por meio da invocação de métodos. Tal como a métrica CTA, isso proporciona uma visão do nível de acoplamento da classe.

2.6 Métricas de Mishra

Mishra [2011] propõe duas métricas para analisar herança em softwares orientados por objetos:

- *Complexidade da Classe pela Herança* (CCI): esta métrica é o somatório da complexidade dos métodos da classe avaliada, acrescida do somatório das medições de CCI de todas suas superclasses. A métrica CCI visa ser um indicador da complexidade que a classe analisada está recebendo das suas superclasses. Segundo Mishra [2011], valores altos indicam classes mais complexas e, consequentemente, mais sujeitas a falhas, devendo receber maior esforço de testes. O principal diferencial desta métrica é que outras métricas de herança propostas na literatura consideram em suas definições a quantidade de classes herdadas, e não a complexidade herdada por meio da hierarquia de herança.

- *Complexidade Média pela Herança* (ACI): esta métrica é uma média de todas as medições de CCI das classes do software. O autor da métrica avalia que softwares com altos valores de ACI possuem uma modelagem complexa e mais sujeita a falhas, enquanto valores baixos de ACI sugerem que o recurso da herança está sendo pouco utilizado no projeto.

2.7 Métricas de Malik & Chhillar

Malik & Chhillar [2011] propõem quatro métricas no nível de classe para avaliar e gerenciar a qualidade de softwares orientados por objetos:

- *Complexidade dos Membros da Classe* (CMCM): esta métrica é o somatório da quantidade de atributos e métodos públicos e protegidos da classe. Segundo Malik & Chhillar, valores altos dessa métrica indicam que o projeto fornece acesso direto a muitos atributos e métodos das classes, o que indica baixo encapsulamento.

- *Complexidade da Herança da Classe* (CICM): esta métrica indica a complexidade introduzida nas classes por meio da herança. Ela é constituída da soma da quantidade de classes pai da classe analisada, acrescida do somatório das medições de CICM de cada uma dessas classes pai. Essa métrica produz os mesmos resultados que a métrica NAC proposta por Li [1998], sendo que ambas divergem da métrica DIT proposta por Chidamber & Kemerer [1994] somente em projetos que utilizam herança múltipla. Projetos que não utilizam herança múltipla irão produzir os mesmo resultados para as três métricas.

- *Nível de Acoplamento da Classe* (CALM): esta métrica indica o nível de acoplamento da classe. Ela é uma razão-proporção entre o número de atributos definidos como tipos de outras classes e o número total de atributos definidos na classe. Segundo Malik & Chhillar, quanto maior o valor dessa métrica, mais acoplada é a classe.

- *Coesão da Classe* (CCOM): é uma métrica para avaliar a coesão da classe, baseada no conceito de compartilhamento de atributos entre os métodos da classe. Esse mesmo conceito é utilizado na definição da métrica LCOM proposta por Chidamber & Kemerer [1994]. A métrica é dada pela razão do somatório da quantidade de métodos que utilizam cada um dos atributos da classe e o número de métodos multiplicado pelo número de atributos. O denominador da razão representa a situação em que cada um dos métodos utiliza todos os atributos da classe. Para os autores, valores próximos de 1 indicam alto grau de coesão na classe.

2.8 Métricas de Rede

Um software orientado por objetos pode ser modelado por meio de um grafo direcionado, no qual um nó representa uma classe do software e uma aresta representa uma relação entre as classe. Essa relação pode ser das seguintes formas:

- Cliente-Servidor: uma classe `A` utiliza serviços providos por uma classe `B`, ou seja, no grafo, `A` e `B` são representados por dois nós e uma aresta direcionada de `A` para `B` representaria a relação entre elas.

- Herança: uma classe `A` é herdeira de uma classe `B`, ou seja, no grafo, `A` e `B` são representados por dois nós e uma aresta direcionada de `A` para `B` representa a relação de herança.

Segundo Newman [2003], esse modelo pode ser enxergado como uma rede complexa, na qual métricas de rede podem ser utilizadas para avaliar o software orientado por objetos. Nesse contexto, o autor apresenta algumas métricas de redes complexas que podem ser aplicadas para o estudo das características desses softwares, dentre elas:

- Grau: esta métrica corresponde ao número de arestas conectadas a um determinado vértice. Esse valor é dado pelo número de arestas, e não pelo número de vértices, haja visto que pode haver mais de uma conexão entre esses dois vértices. Em um grafo direcionado, um vértice possui tanto grau de entrada, como grau de saída. Altos valores para o grau de entrada e saída das classes indicam um projeto com alto grau de acoplamento entre os módulos. O grau de entrada corresponde ao número de conexões aferentes.

- Diâmetro: para determinar o diâmetro de uma rede, deve-se encontrar o caminho mínimo entre cada par de vértices no grafo. Esse caminho mínimo entre cada par de vértices no grafo é chamado de *Geodesic Path*. O maior dentre os *Geodesic Path* encontrados no grafo é o diâmetro desse grafo. Um estudo realizado por Ferreira [2011] identificou que o diâmetro de uma rede de software em geral é curto e cresce pouco à medida que o software cresce, o que sugere que uma modificação em uma classe pode ser facilmente propagada pelo sistema.

- Coeficiente de Clusterização: é uma medida da tendência de que elementos presentes na rede estejam associados a um determinado grupo. Coeficiente de Clusterização é definida da seguinte forma: suponha que um vértice v da rede tenha k_v vizinhos. Podem existir, no máximo, $k_v(k_v - 1)/2$ conexões entre os k_v vizinhos, o que ocorre quando todos os vizinhos estão conectados entre si. Seja C_v uma razão-proporção da quantidade de conexões existentes entre os vizinhos de v e a quantidade total de conexões possíveis entre os vizinhos de v. A métrica C é a média dos C_v para todos os vértices v da rede.

2.9 Sumário das Métricas Estudadas

Nesta seção, sumarizamos na Tabela 5.19 as métricas estudadas, relacionando os autores às métricas propostas por eles. Para cada uma das métricas propostas foram

consideradas as seguintes características: sigla, descrição, escopo e aspecto. O escopo classifica a métrica em termos do que ela avalia: método, classe, pacote ou sistema. O aspecto refere-se à principal característica de qualidade interna avaliada pela métrica.

Tabela 2.2: Sumário das métricas estudadas

Autor(es)	**Métrica**		**Avaliação**	
	Sigla	**Descrição**	**Escopo**	**Aspecto**
Chidamber & Kemerer [1994]	WMC	*Métodos Ponderados por Classe*	Classe	Complexidade
	DIT	*Profundidade de Árvore de Herança*	Classe	Herança
	NOC	*Número de Filhos*	Classe	Herança
	CBO	*Acoplamento entre Classes de Objetos*	Classe	Acoplamento
	RFC	*Resposta de Classe*	Classe	Acoplamento
	LCOM	*Ausência de Coesão em Métodos*	Classe	Coesão
Brito e Abreu & Carapuca [1994]	MHF	*Fator Ocultação de Método*	Sistema	Encapsulamento
	AHF	*Fator Ocultação de Atributo*	Sistema	Encapsulamento
	MIF	*Fator Herança de Método*	Sistema	Herança
	AIF	*Fator Herança de Atributo*	Sistema	Herança
	COF	*Fator Acoplamento*	Sistema	Acoplamento
	PF	*Fator Polimorfismo*	Sistema	Polimorfismo
	RF	*Fator Reúso*	Sistema	Reúso
Martin [1994]	AC	*Acoplamento Aferente*	Pacote	Acoplamento
	EC	*Acoplamento Eferente*	Pacote	Acoplamento
	I	*Instabilidade*	Pacote	Acoplamento
	A	*Abstração*	Pacote	Acoplamento
	RMD	*Distância Normalizada*	Pacote	Acoplamento
[Lorenz & Kidd, 1994]	SIX	*Índice de Especialização*	Classe	Polimorfismo
[McCabe, 1976]	McCabe	*Complexidade de McCabe*	Método	Complexidade
Li [1998]	NAC	*Número de Classes Ancestrais*	Classe	Herança
	NDC	*Número de Classes Descendentes*	Classe	Herança
	NLM	*Número de Métodos Locais*	Classe	Coesão
	CMC	*Complexidade dos Métodos da Classe*	Classe	Complexidade
	CTA	*Acoplamento por Dados Abstratos*	Classe	Acoplamento
	CTM	*Acoplamento por Troca de Mensagens*	Classe	Acoplamento
Mishra [2011]	CCI	*Complexidade da Classe pela Herança*	Classe	Herança
	ACI	*Complexidade Média pela Herança*	Sistema	Herança
Malik & Chhillar [2011]	CMCM	*Complexidade dos Membros da Classe*	Classe	Encapsulamento
	CICM	*Complexidade da Herança da Classe*	Classe	Herança
	CALM	*Nível de Acoplamento da Classe*	Classe	Acoplamento
	CCOM	*Nível de Coesão da Classe*	Classe	Coesão

2.10 Análise Crítica

Os conjuntos de métricas propostos por Chidamber & Kemerer [1994] e Brito e Abreu & Carapuca [1994] são os mais referenciados na literatura. Enquanto o conjunto de métricas CK avalia o software orientado por objetos no nível de classe, o conjunto de métricas MOOD avalia o software como um todo. As métricas CK avaliam aspectos de coesão, acoplamento e herança. As métricas MOOD quantificam, além dos aspectos avaliados nas métricas CK, encapsulamento, polimorfismo e reúso. A utilização dessas

métricas fornecem indicadores de aspectos essenciais para softwares orientados por objeto.

Alguns trabalhos recentes, tais como os propostos por Lamrani et al. [2011] e Lamrani et al. [2013], colaboram com os múltiplos esforços para introduzir as métricas de software orientado por objetos na indústria, tentando formalizar e corrigir ambiguidades nas métricas propostas por Chidamber & Kemerer [1994] e Brito e Abreu & Carapuca [1994]. O trabalho de Li [1998] propõe novas métricas baseadas no trabalho de Chidamber & Kemerer [1994], sanando algumas deficiências que o autor aponta no conjunto CK.

As métricas para avaliação de herança propostas por Mishra [2011] inovam no sentido de capturar a complexidade herdada pelos métodos das classes herdadas. DIT e NAC preocupam-se em medir a quantidade de níveis e classes na hierarquia de herança da classe analisada até a classe raiz. Essas métricas podem ser complementares no sentido de avaliar a herança em softwares orientados por objetos.

Métricas mais recentes como CCI e ACI propostas por Mishra [2011], bem como as métricas CMCM, CICM, CALM e CCOM propostas por Malik & Chhillar [2011], e até mesmo métricas não tão recentes como as métricas propostas Li [1998], dada a pouca disseminação, não possuem ferramentas de coletas de valores tais como as tradicionais métricas CK e MOOD. Isso dificulta a utilização dessas métricas tanto na indústria como em pesquisas científicas que venham a avaliá-las.

As métricas de rede apresentadas têm utilidade na avaliação de aspectos de complexidade da rede formada pelo software, bem como a avaliação dos possíveis impactos em sua complexidade durante o crescimento do software.

O *dataset* utilizado neste trabalho abrange algumas das métricas estudadas no decorrer deste capítulo.

Capítulo 3

Estado da Arte em Valores Referência

Diversas métricas para avaliar aspectos de qualidade em softwares orientados por objetos têm sido propostas na literatura. No entanto, ainda não há indícios de valores considerados como satisfatórios para elas. Esses valores referência poderiam ser utilizados para identificar possíveis problemas no software durante o projeto, implicando em um software de melhor qualidade e consequentemente com um custo de manutenção menor. Saber interpretar corretamente esses valores é de fundamental importância para dar suporte à tomada de decisão dentro do contexto dos projetos de desenvolvimento de software [Baxter et al., 2006; Riaz et al., 2009; Tempero et al., 2010; Ferreira et al., 2012].

Alguns trabalhos 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, 2011; Kaur et al., 2013; Oliveira et al., 2014]. Esses trabalhos variam principalmente na metodologia de pesquisa utilizada para estabelecer esses valores, que, nos trabalhos analisados, podem ser categorizadas da seguinte forma:

- **Pesquisa Quantitativa:** utilizam técnicas estatísticas. Nos trabalhos avaliados estas técnicas são realizadas 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 a boas para identificar a evolução das medições das métricas.

Os trabalhos estudados referentes as categorias de Pesquisa Quantitativa e Qualitativa serão descritos nas Seções 3.1 e 3.2, respectivamente.

3.1 Pesquisa Quantitativa

3.1.1 Regressão Logística

Benlarbi et al. [2000] derivaram empiricamente valores referência para as métricas propostas por Chidamber & Kemerer [1994] via regressão logística, utilizando como amostra dois sistemas de telecomunicações desenvolvidos em C++. Um modelo de regressão logística é uma técnica estatística que explora o relacionamento entre uma variável dependente e uma ou mais variáveis independentes. No trabalho deles, a variável dependente é a falha na classe a as variáveis independentes são os valores das métricas na classe. O objetivo do estudo foi entender como valores referência estabelecidos para as métricas influenciam na variável dependente, que é a falha na classe. Para isso, os autores comparam dois modelos de regressão: um deles utilizando valores referência e o outro não. No modelo que utiliza valores referência, considera-se que não existe probabilidade de haver erro para medições que estejam dentro do valor estabelecido. O objetivo dessa comparação é avaliar a relevância dos valores referência nesse tipo de predição. Os autores concluem que não há diferença significativa entre os modelos para a predição de erro, sendo o modelo que não utiliza o valor referência mais simples, e, consequentemente, preferível.

O trabalho proposto por Benlarbi et al. [2000] apresenta valores referência para 4 métricas, que avaliam o software no nível de classe. Além disso, utiliza-se de uma metodologia complexa para estabelecer e avaliar os valores referência baseada em modelos de regressão. Essa complexidade pode ser um problema para outros estudos que possam vir a complementá-lo. Além disso, os autores utilizam um modelo específico para análise com valores referência, e, como eles mesmo sugerem, seria interessante trabalhar com outros modelos para avaliar os resultados. Como resultado, foi apresentado um conjunto de valores referência para cada um dos dois sistemas utilizados como

amostra e, apesar da conclusão do trabalho, não há uma definição de qual é o valor referência proposto, sendo apresentado dois conjunto de valores referência discrepantes.

Kaur et al. [2013] derivaram empiricamente valores referência para 4 das 6 métricas propostas por Chidamber & Kemerer [1994] via regressão logística. Eles utilizam como amostra duas versões de um mesmo software. O trabalho visa explorar o relacionamento entre a ocorrência de *bad smells* nas classes (variável dependente) e os valores das métricas (variáveis independentes) por meio de um modelo de regressão logística, tal como o trabalho de Benlarbi et al. [2000]. Fowler [1999] define *bad smell* como um problema de projeto em um software, como por exemplo, uma exposição excessiva dos detalhes de implementação de uma classe. No trabalho de Benlarbi et al. [2000], a variável dependente do modelo de regressão logística foi a ocorrência de erro na classe, enquanto que no trabalho de Kaur et al. [2013] a variável dependente foi a ocorrência de *bad smells*. Segundo os autores, os valores referência encontrados são relevantes para identificar classes em risco e que possam apresentar problemas dado os problemas de projeto, demandando, assim, maior esforço de testes. Tal como o trabalho de Benlarbi et al. [2000], são propostos valores referência para métricas que avaliam o software no nível de classe.

3.1.2 Análise de Distribuição

Rosenberg et al. [1999] estabeleceram em seu trabalho valores referência para 7 métricas de forma empírica, com o objetivo de aplicá-las na avaliação da confiabilidade dos softwares na NASA. O estudo foi realizado em mais de 20.000 classes distribuídas em mais de 15 softwares. Especificamente, o objetivo foi identificar valores referência capazes de discriminar códigos chamados de frágeis dos sólidos. Esses valores foram definidos por meio de estudos estatísticos da SATC (*The Software Assurance Technology Center*), realizados com a coleta e análise qualitativa de distribuições estatísticas nos softwares estudados. Os autores não descrevem o tipo de distribuição estatística utilizado. Definidos esses valores, o SATC é capaz de, para cada projeto, criar uma tabela de classes de alto risco. Os autores argumentam que uma única métrica não deve ser utilizada para avaliar os riscos do código, por isso, consideram pelo menos duas métricas para classificar determinada classe como de alto risco.

No trabalho de Rosenberg et al. [1999] é preciso considerar que a análise é feita internamente no contexto de desenvolvimento de software da NASA. Ao concentrar a análise em um domínio de aplicação, ela pode apresentar resultados que não necessariamente podem ser aplicados em outros domínios de aplicação. Contudo, a análise de distribuição estatística dos valores coletados em classes de diversos softwares é uma

metodologia que não apresenta tanta complexidade como a metodologia utilizada em Benlarbi et al. [2000].

Ferreira et al. [2012] identificam por meio da medição das estruturas de um conjunto de 40 softwares de código aberto desenvolvidos em Java, de diferentes tamanhos, domínios e tipos, valores referência para 6 métricas: LCOM, DIT, COF (*coupling factor*), número de conexões aferentes, número de métodos públicos e número de atributos públicos. Foram realizadas 4 análises dos conjuntos de dados: considerando os dados como um todo, domínio de aplicação, tamanho e tipo. Ferreira et al. [2012] concluem que não há diferença significativa entre essas 4 análises, o que é um ponto a ser considerado para outros estudos que visem estabelecer valores referência para outras métricas. O estudo quantitativo apresentado analisa as distribuições dos valores coletados, e conclui que a distribuição desses valores apresenta formato de cauda-pesada. Isso significa que, para a maioria das métricas, há poucas medições com valores altos e grande quantidade de medições com valores baixos. Com base nas distribuições dos valores das métricas encontrados na amostra analisada, os autores propõem valores referência para as métricas. Destaca-se nesse trabalho, um aspecto importante, não considerado nos demais trabalhos pesquisados que se propõem a definir valores referência, que é a validação dos valores encontrados, tanto em classes que violam os princípios da orientação por objetos, como em classes que apresentam bons aspectos de projeto.

Ferreira et al. [2011] identificam valores referência para duas métricas: LCOM4 e COR (Coesão de Responsabilidade). LCOM4 é definida com base nos mesmos conceitos de LCOM, contornando alguns de seus problemas. COR é um indicador do número de responsabilidades implementadas por uma classe [Ferreira et al., 2011]. O trabalho tem por objetivo avaliar a utilidade de métricas de coesão na identificação de classes com problemas estruturais. Para avaliar tais métricas, os valores delas foram comparados a avaliações qualitativas realizadas em classes de softwares abertos desenvolvidos em Java, tanto em cenários avaliados qualitativamente como bons como em cenários avaliados qualitativamente como ruins.

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, atribui-se um peso às entidades do software, baseado nas linhas de código. Depois disso, agrega-se o peso das entidades de acordo com os valores das métricas, por exemplo, complexidade de *McCabe* com valor igual a 17 corresponde a 1,458% do código do sistema *Vuze*. Posteriormente, os pesos são normalizados de acordo com os sistemas, garantindo que a soma das porcentagens dos pesos das entidades continue 100% e, novamente, agrega-se o peso das entidades de acordo com os valores das métricas, dessa vez, para todos os sistemas. Por exemplo, complexidade de *McCabe* com valor igual a

17 corresponde a 0,658% do código de 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 2 métricas em nível de classe.

Oliveira et al. [2014] propuseram 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, mas também é natural em haver um número de entidades na "cauda-longa" que não seguem os limites definidos. Por tal motivo, valores referência absolutos devem ser complementados por uma segunda parte de informação, indicando o percentual de entidades em que o limite superior deve ser aplicado. Os valores referência relativos têm o seguinte formato: $p\%$ das entidades devem ter $M \leq k$, aonde M é a métrica de código-fonte, k é o limite superior e p é a porcentagem mínima. O trabalho de Oliveira et al. focou na descrição do método, derivando valores referência para 7 métricas de classe.

3.1.3 Análise ROC

Shatnawi et al. [2010] investigam a utilização de um método de precisão diagnóstica chamado ROC (*Receiver-Operating Characteristic*) para predição de erros em softwares orientados por objetos. Foi utilizado como amostra do estudo três versões do software Eclipse (2.0, 2.1 e 3.0). Inicialmente, é necessário estabelecer uma faixa de possíveis valores referência para as métricas. Para cada um desses valores, é gerada uma tabela de classificação, a partir do conjunto de classes de amostra, denominada *confusion matrix*. A *confusion matrix* determina, a partir de um valor referência possível e do conjunto de classes da amostra, quantas classes geraram alarme corretamente (verdadeiro-positivo) e incorretamente (falso-positivo), e quantas classes não geraram alarme corretamente (verdadeiro-negativo) e incorretamente (falso-negativo). Cada *confusion matrix* gerará um ponto na curva ROC que representa a classificação daquele valor referência. Alterar os valores referência pode aumentar a classificação correta de classes com erro, mas também pode aumentar as classificações falso-positivas. O critério utilizado para escolher o valor referência tenta minimizar as classificações falso-positivas (alarmes falsos) e falso-negativa (alarme não dado).

Shatnawi et al. [2010] avaliam duas hipóteses com a utilização das curvas ROC.

A primeira hipótese é que não há valores referência para as métricas de softwares orientados por objetos que distingam as classes em duas categorias: com e sem erros. A segunda hipótese é que não há valores referência para métricas de softwares orientados por objetos capazes de classificar as classes em alguma das seguintes categorias de erros: erros de baixo risco, erros de médio risco e erros de alto risco. Alternativamente, a classe também pode ser classificada como sem erros. A primeira hipótese não foi rejeitada pela pesquisa, pois a qualidade da classificação não conseguiu chegar a faixa aceitável. A segunda hipótese foi rejeitada, haja visto que a classificação proporcionada chegou a faixa aceitável de qualidade de categorização da curva ROC, para as categorias de erros de médio risco e alto risco. Isso significa que o trabalho identificou valores referência capazes de distinguir classes com erros categorizados em médio risco e alto risco, para 5 métricas: CBO, RFC, WMC, CTM e NOO (*Number of Operations*). Para classes com baixo risco, a classificação não se mostrou eficiente.

O trabalho de Shatnawi et al. [2010] foi aplicado em uma amostra reduzida em termos de domínio e escala. Os valores referência foram derivados para cada uma das três versões do Eclipse avaliadas, apresentando diferenças significativas que não foram justificadas. Para definir os valores referência finais, Shatnawi et al. utilizam aqueles que apresentaram melhor qualidade de classificação dentre os valores derivados para cada um dos softwares.

3.2 Pesquisa Qualitativa

Chhikara et al. [2011] avaliam possíveis modelagens de um mesmo problema e medem valores das métricas de Chidamber & Kemerer [1994], analisando a evolução das medições nessas modelagens, com o intuito de mostrar como a herança é um fator essencial nos sistemas orientados por objetos. O trabalho propõe 4 modelagens de classe para resolver um mesmo problema. As modelagens são alteradas em termos de níveis de herança e variam progressivamente em termos de qualidade, até que o projeto seja considerado ideal. Os autores medem 6 métricas nesses projetos (WMC, DIT, RFC, NOC, CBO e LCOM) e avaliam como as medições das métricas evoluíram com o aprimoramento dos projetos propostos, avaliando assim o impacto da utilização da herança nas métricas.

Apesar do trabalho de Chhikara et al. [2011] não apresentar uma avaliação em grande escala, ele buscou identificar a evolução dos valores das métricas em função da melhoria da qualidade das modelagens, permitindo identificar alguns valores considerados ideais, dado o contexto do problema. No entanto, pela limitação da análise, não

é possível afirmar que esses valores podem ser tomados como base para outros estudos, pela especificidade dos casos analisados.

3.3 Resumo dos Trabalhos Estudados

A Tabela 3.1 sumariza os valores referência encontrados na literatura, agrupados por estudo, para o conjunto CK. Essa mesma sumarização não foi realizada para as outras métricas estudadas pois não há similaridade entre as demais métricas estudadas nos trabalhos que permita essa comparação.

Tabela 3.1: Valores referência para métricas CK na literatura

Autor	WMC	DIT	NOC	CBO	RFC	LCOM
Benlarbi et al. [2000] System 1	11	2	2	1	27	-
Benlarbi et al. [2000] System 2	31	1	-	8	25	-
Rosenberg et al. [1999]	≤ 40	2|3	-	≤ 5	≤ 10	-
Chhikara et al. [2011]	≤ 9	≤ 3	≤ 3	0	≤ 9	≤ 2
Ferreira et al. [2012]	-	≤ 2	-	-	-	0
Kaur et al. [2013]	14	1	-	7	31	-
Shatnawi et al. [2010]	24	-	-	13	44	-
Oliveira et al. [2014]	$80\% \leq 32$	-	-	-	$80\% \leq 49$	$80\% \leq 36$
Alves et al. [2010]	-	-	-	-	-	-

3.4 Análise Crítica

A contribuição principal deste trabalho de dissertação é um catálogo de valores referência para 18 métricas de softwares orientados por objetos, que cobre uma quantidade maior de métricas, possibilitando a avaliação de métodos, classes e pacotes. Pelo exposto no Capítulo 3, percebe-se que, embora existam pesquisas anteriores que propõem diferentes técnicas para derivar valores referência para métricas de softwares orientados por objetos, a maioria deles cobre poucas métricas. Nesse cenário, não pretendemos propor um novo método para derivação de valores referência. Ao invés disso, aplicamos o método empírico proposto por Ferreira et al. [2012] na identificação dos valores. Além disso, introduzimos algumas melhorias nesse método com o objetivo de avançar nas definições e técnicas estatísticas utilizadas. Quando as contribuições deste trabalho são comparadas com os resultados apresentados por Ferreira et al. [2012] e outros estudos anteriores, é possível identificar duas diferenças principais. (1) Foram identificados valores referência para um número maior de métricas. (2) Os valores referência propostos provêem um *benchmark* para a avaliação quantitativa da qualidade interna

de softwares orientados por objetos, considerando não somente classes, mas também métodos e pacotes.

Em relação aos métodos propostos, percebe-se que as pesquisas relacionadas com valores referência para as métricas de software orientado por objetos vem sendo realizadas com metodologias de pesquisa distintas. As pesquisas realizadas por Benlarbi et al. [2000] e Kaur et al. [2013] utilizam regressão logística para definir os valores referência para métricas de software orientado por objetos. Essa metodologia de pesquisa é complexa, o que dificulta sua aplicação. Como pode ser observado pela Tabela 3.1, os valores encontrados em Benlarbi et al. [2000] no primeiro sistema analisado e Kaur et al. [2013] são semelhantes, o que sugere que os estudos que se utilizam da mesma metodologia podem obter resultados semelhantes. No entanto, há uma discrepância entre os valores coletados nos dois sistemas analisados em Benlarbi et al. [2000], principalmente nas métricas WMC e CBO. Os autores não avaliam o porquê dessa discrepância em sua pesquisa.

A pesquisa realizada por Shatnawi et al. [2010] utiliza um método de precisão diagnóstica chamado ROC para definir valores referência capazes de classificar classes em situações de erro. Essa metodologia de pesquisa é complexa e depende de uma base de registro de classes em situações categorizadas de erros. Esse tipo de informação não é comum, o que dificulta a aplicação dessa metodologia em larga escala. Essa é, inclusive, uma das limitações do trabalho citadas pelos autores, que utilizaram dos registros de *log* do projeto Eclipse para coletar essas informações. Como o registro é manual, algum erro que não tiver sido registrado não será considerado no estudo.

Tal como o trabalho de Shatnawi et al. [2010], os trabalhos de Benlarbi et al. [2000] e Kaur et al. [2013], que necessitam de uma variável dependente para aplicar a regressão logística, têm o mesmo problema. O trabalho de Benlarbi et al. [2000] foi aplicado em dois softwares que possuíam a informação de registro de erros das classes. O trabalho de Kaur et al. [2013] foi aplicado em duas versões de um mesmo software que possuíam a informação de *bad smells* de cada uma das classes. Isso demonstra que estudos que associam a definição das métricas a uma determinada categorização possuem a dificuldade de depender de amostras já categorizadas, o que pode limitar a escala da aplicação do método.

As pesquisas de Rosenberg et al. [1999] e Ferreira et al. [2012] utilizam de coleta de valores em grandes quantidades de software e de análise de distribuição estatística para derivar os valores referência. Essa metodologia de pesquisa é mais clara e de melhor entendimento do que a utilização de regressão logística. No entanto, não foi possível identificar no trabalho de Rosenberg et al. [1999] o tipo de distribuição que foi utilizado na definição dos valores. Isso é de fundamental importância, como exposto em

Ferreira et al. [2012]. Como pode ser observado pela Tabela 3.1, o valor referência para a métrica DIT encontrado no trabalho Ferreira et al. [2012] tem valor semelhante ao valor encontrado por Rosenberg et al. [1999]. Ferreira et al. [2012] não trabalham com as outras métricas CK, o que impossibilita uma comparação mais ampla dos resultados dos dois trabalhos, que utilizam de metodologias de pesquisa semelhantes. É possível observar que o trabalho de Rosenberg et al. [1999] tem resultados semelhantes aos trabalhos de Benlarbi et al. [2000] e Kaur et al. [2013], exceto no caso da métrica RFC.

No trabalho de Alves et al. [2010], apesar dos valores referência conterem o termo "risco", o trabalho não apresenta uma avaliação da validade dos perfis de qualidade estabelecidos. Além disso, a adoção de percentis fixos para particionar os perfis de qualidade sugeridos não funciona, necessariamente, para todas as métricas, afinal, há de se levar em conta as características acerca de como os dados se distribuem. Devido a esta limitação, no presente trabalho, realizamos a análise dos dados por meio da compreensão da curva de distribuição dos valores para estabelecer esses percentis, podendo variar de métrica para métrica. Além disso, acreditamos que quatro perfis de qualidade de risco trazem uma complexidade desnecessária aos valores referência, principalmente na distinção entre alto risco e altíssimo risco.

A abordagem proposta por Oliveira et al. [2014] possui propósitos válidos em alguns contextos, como no cenário visualizado para medir o débito técnico [Cunningham, 1992] em um sistema. Nos valores referência propostos neste trabalho de pesquisa, quando uma entidade de código é mal avaliada pelo valor referência (faixa *Ruim/Raro*), acreditamos que ela deva ser submetida à inspeção. Por exemplo, se existirem dois métodos com 35 linhas de código em um sistema, ambos deveriam ser inspecionados e possivelmente refatorados. Na abordagem proposta por Oliveira et al., a refatoração de apenas um deles pode levar o sistema a um patamar de qualidade aceitável dentro do contexto do controle e gerenciamento do débito técnico do sistema. Portanto, acreditamos que a abordagem proposta por Oliveira et al. (valores referência relativos) e a abordagem adotada neste trabalho (valores referência absolutos) não são excludentes, porém, complementares nos esforços correntes em aplicar de forma efetiva as métricas no gerenciamento da qualidade de software.

Chhikara et al. [2011] utilizam uma metodologia de pesquisa qualitativa por meio da análise de possíveis projetos que variam em qualidade para um mesmo problema. Essa análise é restritiva, principalmente em termos de porte do software. No entanto, percebe-se que, apesar da metologia de pesquisa possuir menos complexidade em relação àquelas aplicadas nos demais trabalhos, os valores das métricas CK estudadas, DIT e NOC, são semelhantes àqueles identificados nos demais estudos. Da mesma forma, o valor referência encontrado para RFC é semelhante ao identificado por Rosenberg

et al. [1999]. A métrica WMC apresenta valor referência inferior aos demais trabalhos.

Percebe-se, pelos estudos avaliados, que há uma concentração mais acentuada da pesquisa em valores referência para as métricas CK. Outras métricas propostas na literatura não foram exploradas nesse tipo de pesquisa. Este fato mostra que há um nicho de pesquisa em aberto em relação a definição de valores referência para métricas de software orientado por objetos. Neste trabalho, ampliamos o universo de métricas com valores referência derivados para fora do conjunto CK.

Outro ponto a ser considerado é a validação dos valores referência propostos. Avaliar a eficácia da utilização dos valores referência propostos na identificação de classes de boa e má qualidade é um ponto que deve ser cuidadosamente analisado. Em sua maior parte, os valores referência propostos não foram amplamente avaliados, o que é fundamental para a aplicá-los na prática. O trabalho de Rosenberg et al. [1999] descreve brevemente a utilização dos valores referência no SATC (The Software Assurance Technology Center), mencionando a utilização das tabelas definidas a partir dos valores referência na tomada de decisões gerenciais. No entanto, o trabalho não descreve os resultados obtidos com a adoção desse processo. O trabalho de Ferreira et al. [2012] avalia os valores derivados em classes bem e mal modeladas, concluindo que os valores propostos podem ser utilizados com esse propósito.

3.5 Conclusão

Diante do exposto neste capítulo e considerando que há um grande campo de pesquisa em aberto na definição de valores referência para métricas de software orientado por objetos, que os valores referência de diversas métricas propostas na literatura ainda não são conhecidos, este trabalho de pesquisa tem por objetivo contribuir com a solução para esse problema, identificando e avaliando valores referência para um conjunto de 18 métricas de softwares orientados por objetos. Neste sentido propusemos um catálogo cobre uma quantidade maior de métricas do que outros estudos. Além disso, não engloba somente métricas de nível de classe, mas também métricas de nível de métodos e pacotes.

Capítulo 4

Um Método de Extração de Valores Referência

O objetivo principal do trabalho desta dissertação de mestrado é disponibilizar um catálogo de valores referência para métricas de softwares orientados por objetos. Há diversas métricas de sofwares propostas na literatura, mas poucas possuem valores referência indicados. Para contribuir para a solução desse problema, serão identificados valores referência para 18 métricas de software orientado por objetos, permitindo sua utilização na avaliação quantitativa da qualidade interna de softwares dessa natureza. O método aplicado para a condução desta pesquisa é descrito neste capítulo.

4.1 Método

Conforme observado no Capítulo 3, existem diversas abordagens utilizadas na literatura para derivar valores referência para métricas de softwares orientados por objetos. Neste trabalho, será utilizada a abordagem proposta por Ferreira et al. [2012], baseada na análise da distribuição estatística dos valores das métricas. Optamos por essa abordagem pois, conforme observado na Seção 3.4, os trabalhos que se baseiam em Regressão Logística e Análise ROC apresentam alto grau de complexidade estatística, dificultando sua aplicação. Além disso, a necessidade das variáveis dependentes exige amostras categorizadas, o que limita a escala da aplicação desses métodos. O método qualitativo estudado (Seção 3.2) apresenta um problema de validação, pois a amostra é limitada a casos específicos, construídos diretamente para aquele fim. Por outro lado, a abordagem escolhida possui alto grau de escalabilidade e, dentre os métodos estudados, apresenta simplicidade, eficiência e validade de resultados. Este capítulo está organizado da seguinte forma: Seção 4.1.1 apresenta o Qualitas.*class* Corpus, repositó-

rio de sistemas utilizado como *dataset* no trabalho; Seção 4.1.2 descreve a metodologia aplicada na pesquisa para derivar os valores referência, e Seção 4.2 apresenta a forma escolhida para avaliar os valores referência derivados.

4.1.1 Qualitas.*class* Corpus

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. Qualitas Corpus é uma 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 é prover um recurso para que estudos acerca de softwares possam ser reproduzidos. As versões compiladas dos projetos Java na ferramenta Eclipse [2013] incluem os 111 sistemas do *Corpus*[1]. O conjunto de softwares contém mais de 18 milhões de linhas de código, 200.000 classes e 1.5 milhões de métodos compilados. A Tabela 4.1 sumariza os dados gerais do Qualitas.*class* Corpus.

Tabela 4.1: Dados gerais do Qualitas.*class* Corpus. Fonte: Terra et al. [2013]

Chave	**Valor**
Sistemas	111
Linhas de Código	18.548.026
Número de Projetos	802
Númedo de pacotes	16.509
Número de Classes	202.052
Número de Interfaces	22.115
Número de Métodos	1.464.893

O *Corpus* é uma coleção de sistemas, cada um contendo um ou mais projetos. Por exemplo, o sistema AspectJ[2] é composto de quatro projetos internos: *matcher*, *rt*, *tools* e *weaver*. Considerando esse aspecto, o Qualitas.*class* Corpus possui um total de 802 projetos internos, incluindo 406 projetos do NetBeans[3]. No site[4] do projeto de Terra et al. [2013] são disponibilizados 802 arquivos XML referentes a cada um dos 802 projetos que compõem os 111 sistemas disponíveis no Qualitas.*class* Corpus. Cada arquivo contém medições de 23 métricas de softwares.

Para cada projeto P, o Qualitas.*class* Corpus disponibiliza um arquivo XML com elementos *Metric* para cada métrica M, identificada pelo atributo id.

[1] A versão utilizada foi 20120401
[2] Disponível em: `http://eclipse.org/aspectj`
[3] Disponível em: `http://netbeans.org`
[4] `http://java.labsoft.dcc.ufmg.br/qualitas.class`

Um exemplo de arquivo XML é mostrado a seguir, e corresponde ao projeto *batik-1.7_samples*. O arquivo foi reduzido com o objetivo de ilustrar a estrutura dos arquivos disponibilizados no trabalho de Terra et al. [2013].

```
1<Metrics scope="batik-1.7_samples">
2 <Metric id="VG">
3  <Values per="method">
4   <Value name="run" source ="UntrustedScriptHandler.java"
        package ="com.untrusted.script" value ="6"/>
5  </Values>
6 </Metric>
7 <Metric id="CE">
8  <Values per="packageFragment">
9   <Value name="com.test.script" package="com.test.script"
        value="1" />
10  </Values>
11 </Metric>
12 <Metric id="DIT">
13  <Values per="type">
14   <Value name="EventListenerInitializerImpl" source="
        EventListenerInitializerImpl.java" package="com.test.
        script" value="1" />
15  </Values>
16 </Metric>
17 <Metric id = "NOP" description ="Number of Packages">
18         <Value value="2"/>
19    </Metric>
20</Metrics>
```

Na *linha 1*, percebe-se o elemento *Metrics*, identificado pelo atributo *scope*, que é o nome do projeto. Esse elemento é composto por n elementos filhos do tipo *Metric*, que contém o atributo *id* que identifica a métrica. Por exemplo, na *linha 2*, a métrica reportada é a complexidade de *McCabe*, identificada por $id = VG$. Na *linha 7*, a métrica reportada é o acoplamento eferente, identificada por $id = EC$. Na linha 12, a métrica reportada é a profundidade de árvore de herança, identificada por $id = DIT$.

Cada elemento do tipo *Metric* é composto por um elemento filho do tipo *Values*. Esse elemento do tipo *Values* contém um atributo denominado *per*, que identifica se aquela métrica é dada no escopo de método (*method*, na *linha 3*), classe (*type*, na *linha*

13) ou pacote (*packageFragment*, na *linha 8*).

O elemento do tipo *Values* é composto por n elementos do tipo *Value*, representando a medida em si. Esse elemento *Value* é composto dos atributos *name*, *source*, *package* e *value*. Os casos de medições por método, classe ou pacote são realizados da seguinte forma:

- método: o atributo *name* especifica o nome do método medido, *source* o nome do arquivo da classe em que o método se encontra, e *package* o pacote em que a classe se encontra. Um exemplo pode ser visto na *linha 4*;
- classe: o atributo *name* especifica o nome da classe, *source* o nome do arquivo da classe, e *package* o pacote em que a classe se encontra. Um exemplo pode ser visto na *linha 14*;
- pacote: os atributos *name* e *package* especificam o nome do pacote. O atributo *source* não é utilizado. Um exemplo pode ser visto na *linha 9*.

Os casos especificados possuem o atributo *value*, que é a medida da métrica. Há ainda medições que se referem ao sistema como um todo, não possuindo distinção entre método, classe ou pacote. É o caso das métrica número de pacotes (NOP) e total de linhas de código (TLOC). Conforme pode ser visto na *linha 17*, NOP contém apenas o elemento *Value* com o atributo *value* especificado.

Terra et al. [2013] agrupam as 23 métricas da seguinte forma:

- **Métricas Básicas:** linhas de código por método (MLOC), número de pacotes (NOP), número de classes (NOC), número de interfaces (NOI), número de métodos (NOM), número de atributos (NOF), número de métodos sobrescritos (NORM), número de parâmetros (PAR), número de métodos estáticos (NSM) e número de atributos estáticos (NSF).
- **Métricas de Complexidade:** número de linhas de código do métodos (MLOC), índice de especialização (SIX), complexidade de *McCabe* (VG), profundidade de blocos aninhados (NBD) e distância normalizada (RMD).
- **Conjunto CK**: métodos ponderados por classe (WMC), profundidade de árvore de herança (DIT), número de filhas (NSC) e ausência de coesão em métodos (LCOM).
- **Métricas de Acoplamento:** acoplamento aferente (CA), acoplamento eferente (CE), instabilidade (I) e abstração (A).

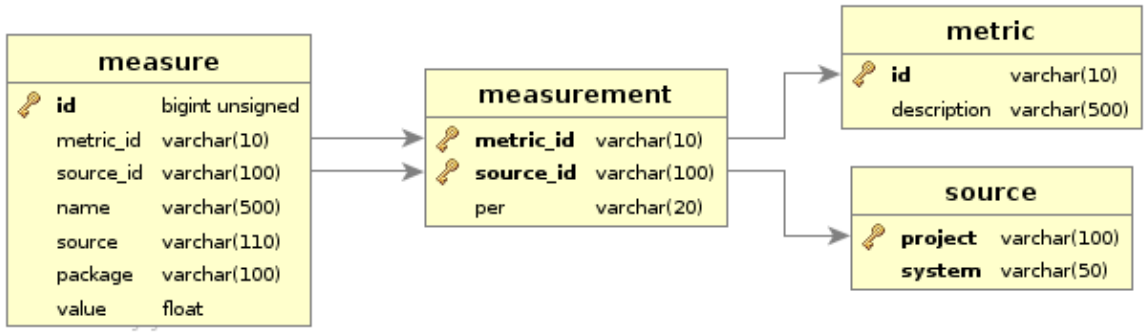


Figura 4.1: Diagrama ER - Armazenamento das medidas, medições, métricas e projetos disponibilizados no Qualitas.*class* Corpus

Neste trabalho, utilizam-se 18 das 23 métricas coletadas no trabalho de Terra et al. [2013] para aplicar a metodologia proposta no trabalho de Ferreira et al. [2012], com o objetivo de derivar valores referências para essas métricas. Foram excluídas as métricas TLOC e NOP, por serem medidas únicas dentro dos projetos de software, não propiciando a realização de uma análise estatística apropriada. As métricas I e A também foram excluídas da análise, pois são métricas cujas medidas são dependentes para a avaliação da qualidade, não sendo possível indicar um valor que expresse uma boa ou má qualidade.

4.1.2 Método de Derivação dos Valores Referência

Nesta seção é descrita a metodologia aplicada na derivação de valores referência para métricas de softwares orientados por objetos.

4.1.2.1 Preparaçao dos Dados

Para a realização deste trabalho, desenvolvemos uma ferramenta que lê os arquivos XML disponibilizados no Qualitas.*class* Corpus utilizando o JAXB[5]. JAXB provê uma forma rápida e fácil de ler documentos XML, realizando o processo de *Unmarshall*, que é a conversão do documento XML para uma árvore de objetos Java. A partir dos objetos, a ferramenta alimenta um banco de dados *Mysql* com o objetivo de estruturar e normalizar as medidas.

O modelo de dados proposto pode ser visto por meio do diagrama entidade relacionamento (DER) na Figura 4.1. A Tabela `metric` armazena as 18 métricas cujas medições foram realizadas no Qualitas.*class* Corpus. A Tabela `source` representa cada um dos 802 projetos disponibilizados no estudo. A Tabela `measurement` representa

[5] `http://docs.oracle.com/javase/tutorial/jaxb`

o ato de medir uma métrica para um projeto, por isso, ela aponta para as Tabelas `metric` e `source`. Uma medição possui várias medidas a ela relacionadas. Por exemplo, a medição da métrica WMC em determinado projeto gera uma medida para cada método. Por isso, a Tabela `measure`, que armazena as medidas realizadas para cada medição, aponta para a Tabela `measurement`.

Realizada a conversão por meio da ferramenta desenvolvida, tem-se uma forma rápida e ágil de agregar e sumarizar os dados. Por exemplo, a tabela de frequência absoluta da métrica WMC pode ser facilmente obtida. Uma tabela de frequência absoluta conta o número de vezes que determinada medida ocorreu. O seguinte comando SQL obtém essa tabela no banco de dados:

```
1 SELECT
2   value, count(value)
3 FROM
4   measure
5 WHERE
6   metric_id = 'WMC'
7 GROUP BY value;
```

A Tabela 4.2 sumariza a quantidade de medidas obtidas por métrica, que serão utilizadas nas fases posteriores do trabalho para derivar os valores referência.

4.1.2.2 *Data-Fitting*

Segundo Gibbons & Chakraborti [2003], um problema importante na estatística é obter informação sobre a distribuição seguida pelo conjunto de dados analisados. Para esse propósito, a ferramenta chamada EasyFit [2013], por meio do teste de aderência de *Kolmogorov-Smirnov*, realiza a seleção da distribuição apropriada aos dados. Selecionar a distribuição apropriada aos dados é o procedimento de escolher uma distribuição que tenha melhor ajuste para um conjunto de dados.

O objetivo dessa etapa da pesquisa é estabelecer a distribuição apropriada a cada métrica de software estudada. Segundo Baxter et al. [2006], explorar as distribuições dos valores das métricas de software é fundamental para avançar no entendimento das estruturas internas de softwares. A partir da distribuição, é possível identificar e entender suas características, por exemplo, se o valor médio é ou não representativo para a análise.

EasyFit recebe como entrada as medidas coletadas por métrica. Elas podem ser obtidas por meio de um comando SQL no banco de dados proposto na Seção 4.1.2.1. Por exemplo, para a métrica WMC:

Tabela 4.2: Quantidade de medidas por métrica.

Id	Métrica	Contagem
CA	Acoplamento Aferente	16.566
CE	Acoplamento Eferente	16.566
DIT	Profundidade de Árvore de Herança	247.395
LCOM	Ausência de Coesão em Métodos	247.395
MLOC	Linhas de Código por Método	1.663.248
NBD	Profundidade de Blocos Aninhados	1.663.248
NOC	Número de Classes	16.566
NOF	Número de Atributos	247.395
NOM	Número de Métodos	247.395
NORM	Número de Métodos Sobrescritos	247.395
NSC	Número de Filhas	247.395
NSF	Número de Atributos Estáticos	247.395
NSM	Número de Métodos Estáticos	247.395
PAR	Número de Parâmetros	1.663.248
RMD	Distância Normalizada	16.566
SIX	Índice de Especialização	247.395
VG	Complexidade de *McCabe*	1.663.248
WMC	Métodos Ponderados por Classe	247.395

```
1 SELECT value
2 FROM measure
3 WHERE metric_id = 'WMC';
```

Como saída, *EasyFit* define a distribuição apropriada aos dados.

4.1.2.3 Gráficos Utilizados

Foram utilizados sete tipos de gráficos como ferramentas para as análises realizadas no processo de derivação das métricas. São eles: gráficos de dispersão e de frequência relativa acumulada, funções densidade de probabilidade e de distribuição cumulativa e histograma de probabilidade e histograma em escala *log-log*. Esses gráficos representam o comportamento estatístico de cada uma das métricas estudadas, auxiliando na descrição dos dados coletados. O entendimento desses gráficos é de suma importância para compreender a forma como os dados se distribuem, fundamentando o processo de derivação. A ferramenta R[6] foi utilizada para geração dos gráficos de dispersão, frequência relativa acumulada, histograma de probabilidade e histograma em escala *log-log*, enquanto que para a geração dos gráficos de função de densidade de probabilidade e

[6] http://www.r-project.org/

de distribuição cumulativa ajustada à distribuição sugerida, utilizou-se a ferramenta *EasyFit*. Os tipos de gráficos utilizados neste trabalho são descritos a seguir.

- **Gráfico de Dispersão:** de acordo com R [2014b], um gráfico de dispersão pareia valores de duas variáveis quantitativas em um conjunto de dados, exibindo-os como pontos geométricos dentro de um diagrama cartesiano. No contexto deste trabalho, pareiam-se os valores da métrica (eixo x) com a contagem de ocorrências desses valores (eixo y). Como exemplo, seja o gráfico de dispersão da métrica Acoplamento Aferente (AC) ilustrado na Figura 4.2. O valor $x = 0$, que representa o valor da métrica, possui contagem de ocorrências $y = 5255$.

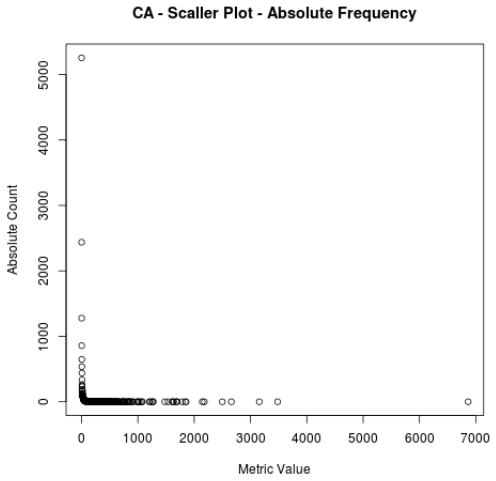


Figura 4.2: Acoplamento Aferente: Gráfico de Dispersão

- **Gráfico de Frequência Relativa Acumulada:** de acordo com R [2014b], um gráfico de frequência relativa acumulada de uma variável quantitativa é uma curva gráfica que mostra a frequência relativa acumulada da distribuição. A frequência relativa acumulada de uma distribuição de uma variável quantitativa é um resumo da proporção de frequência abaixo de um determinado nível. No contexto deste trabalho, pareiam-se os valores da métrica (eixo x) com a frequência relativa acumulada abaixo daquele valor (nível). Como exemplo, seja o gráfico de frequência relativa acumulada da métrica Acoplamento Aferente (CA) ilustrado na Figura 4.3. O valor $x = 2$, que representa o valor da métrica, possui frequência relativa acumulada abaixo daquele valor $y = 0,46$. Isso significa que, abaixo de 2, tem-se 46% dos dados do conjunto.

- **Função Densidade de Probabilidade ajustada à distribuição sugerida:** segundo EasyFit [2013], a função densidade de probabilidade (*pdf*) é a probabili-

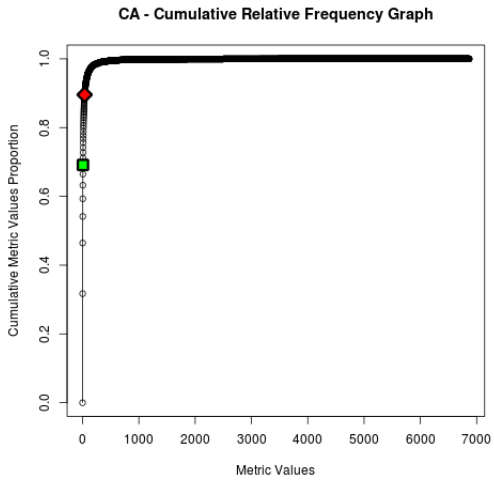


Figura 4.3: Acoplamento Aferente: Gráfico de Frequência Relativa Acumulada

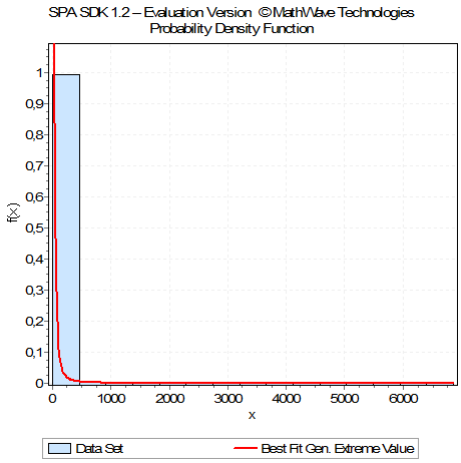


Figura 4.4: Acoplamento Aferente: Função de Densidade de Probabilidade ajustada à Distribuição Generalized Extreme Value

dade de uma variável assumir um valor x: $f(x) = p(X = x)$. É um gráfico similar ao histograma de probabilidade, que contém, além das barras que expressam a probabilidade da variável assumir determinado valor, uma linha que corresponde ao ajuste dos dados à distribuição sugerida. A Figura 4.4 ilustra o gráfico da função densidade de probabilidade da métrica Acoplamento Aferente (AC).

- **Função de Distribuição Cumulativa ajustada à distribuição sugerida:** segundo EasyFit [2013], a função de distribuição cumulativa (cdf) é a probabilidade de uma variável assumir um valor menor ou igual x: $F(x) = p(X \leq x)$. É um gráfico similar ao gráfico de frequência relativa acumulada, que contém, além da probabilidade da variável assumir um valor menor ou igual que a variável, uma

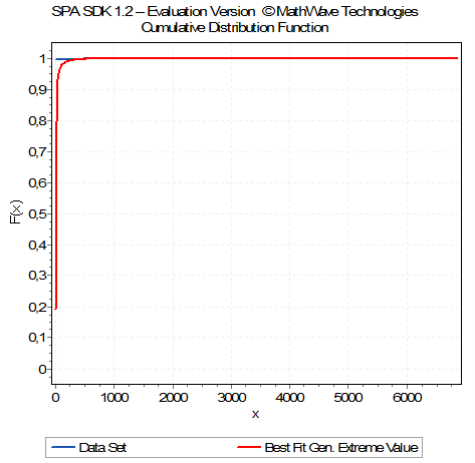


Figura 4.5: Acoplamento Aferente: Gráfico de Frequência Relativa Acumulada ajustado à Distribuição Generalized Extreme Value

linha que corresponde ao ajuste dos dados à distribuição sugerida. A Figura 4.5 ilustra o gráfico da função de distribuição cumulativa da métrica Acoplamento Aferente (AC).

- **Histograma de Probabilidade:** de acordo com R [2014b], um histograma consiste em barras verticais paralelas que, graficamente, exibem a distribuição da frequência de uma variável quantitativa. O conjunto de variáveis é agrupado em classes. Cada classe é um retângulo cuja base horizontal é o intervalo correspondente àquela classe (eixo x). A altura vertical da barra representa a frequência com que os valores da classe acontecem. No histograma de probabilidade, a altura da barra representa a frequência relativa da ocorrência da classe, ou seja, ao invés de apresentar o valor absoluto, apresenta a porcentagem da frequência absoluta em relação à quantidade total dos dados. No contexto deste trabalho, as classes são formadas pelos intervalos correspondentes aos valores da métrica (eixo x) e a altura vertical da barra é a frequência relativa daquela classe. Como exemplo, seja o histograma de probabilidade da métrica Acoplamento Aferente (CA) ilustrado na Figura 4.6. O valor $x = [20, 40)$, que representa os valores da métrica correspondentes àquela classe, possui frequência relativa $y = 0,14$. Isso significa que, para classes constituídas dos valores de CA entre 20 e 40, tem-se 15% dos dados do conjunto.

- **Histograma em escala *log-log*:** de acordo com R [2014b], um histograma consiste de barras verticais paralelas que exibem de forma gráfica a distribuição de frequência de uma variável quantitativa. O histograma em escala *log-log* exibe

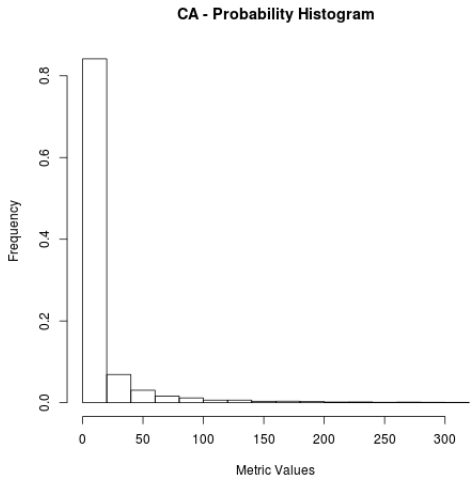


Figura 4.6: Acoplamento Aferente: Histograma de Probabilidade

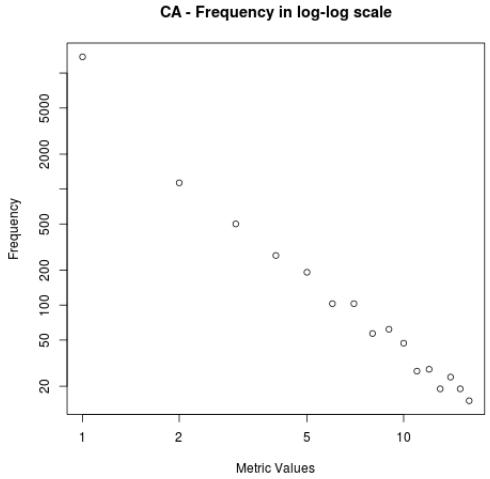


Figura 4.7: Acoplamento Aferente: Histograma em escala *log-log*.

esses mesmos dados em escala logarítmica, com o objetivo de observar o comportamento da curva gerada e entender como os dados analisados se distribuem. A Figura 4.7 ilustra o histograma em escala *log-log* da métrica Acoplamento Aferente (AC).

- **Função de Densidade de Probabilidade:** mostra a Função Densidade de Probabilidade da distribuição com os parâmetros retornados, sem o ajuste dos dados da amostra à distribuição sugerida, como acontece no Gráfico Função de Distribuição Cumulativa ajustada à distribuição sugerida. Esse gráfico proporciona uma visão mais clara das características da distribuição sugerida com os respectivos parâmetros. A Figura 4.8 ilustra esse gráfico.

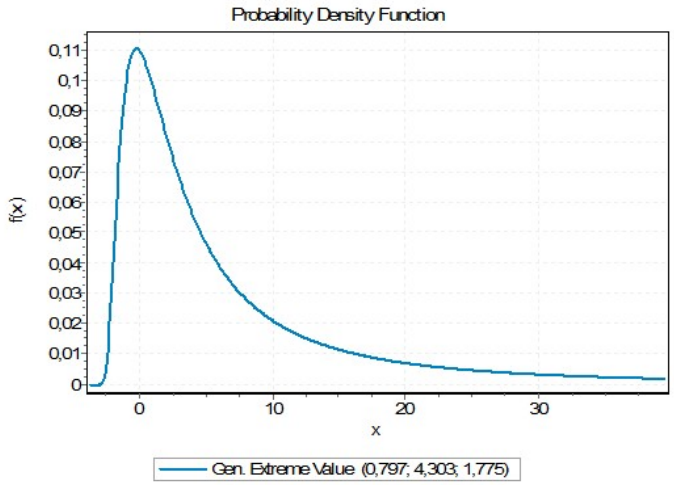


Figura 4.8: CA - Função de Densidade de Probabilidade com os parâmetros retornados pela ferramenta *EasyFit*

4.1.2.4 Análise dos Dados

Para cada métrica, tem-se a distribuição de probabilidade apropriada ao conjunto de medidas (Seção 4.1.2.2) e um conjunto de gráficos que as descrevem (Seção 4.1.2.3). A partir das visualizações gráficas e do conhecimento das características acerca da distribuição de probabilidade apropriada ao conjunto de medidas, é possível derivar valores referência para as métricas.

Na abordagem proposta por Ferreira et al. [2012], quando a métrica apresenta uma distribuição que possui valor médio representativo, como por exemplo, a distribuição de *Poisson*, esse valor representa o valor típico da métrica. Caso contrário, são identificadas três faixas de valores: *bom*, *regular* e *ruim*. A faixa *bom* corresponde a valores com alta frequência, caracterizando os valores mais comuns da métrica, na prática. Segundo Ferreira et al., esses valores não expressem necessariamente as melhores práticas da Engenharia de Software, e sim um padrão seguido pela maioria dos softwares. A faixa *ruim* corresponde a valores com baixa frequência, e a faixa *regular* é intermediária, correspondendo a valores que não são nem muito frequentes nem raros. A frequência é estabelecida por meio de uma análise visual do histograma de probabilidade dos dados ajustados a distribuição.

Neste trabalho de dissertação foram propostas melhorias ao método original proposto por Ferreira et al. [2012]. Primeiramente, os nomes das faixas foram modificados para: *Bom/Frequente*, *Regular/Ocasional* e *Ruim/Raro*. Acredita-se que esses nomes expressam de uma forma mais clara a importância do conceito da frequência nos valores referência identificados.

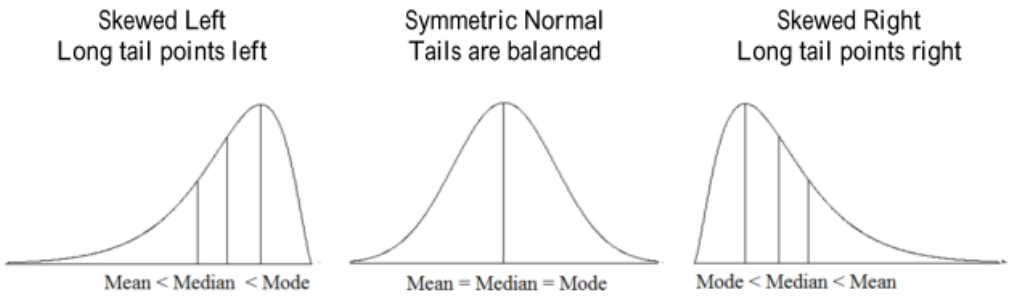


Figura 4.9: Curvas mostrando a posição da média, mediana e moda em relação à simetria da curva. Fonte: Doane & Seward [2011]

Em segundo lugar, em vez de derivar os valores referência diretamente da análise visual, estabeleceu-se, também baseados em uma análise visual dos gráficos plotados, dois percentis, capazes de separar o conjunto de dados em três partes que correspodem às faixas sugeridas para os valores referência. Apesar de a análise visual ainda ser necessária para a aplicação do método, a utilização dos percentis traz uma melhoria relevante ao método original, pois permite a obtenção dos valores referência diretamente do conjunto de dados, com a utilização de ferramentas estatísticas que retornam o valor correspondente ao percentil[7], o que torna a aplicação do método mais reprodutível.

Mesmo que a análise visual ainda se faça necessária, acredita-se que foi proporcionada uma melhor fundamentação visual para a escolha das regiões do que no método original, pois foram usados 7 gráficos enquanto no método original foram utilizados 3.

Uma terceira melhoria no método foi a introdução do conceito da assimetria, que é fundamental quando analisa-se a forma como os dados se distribuem. Dados assimétricos se distribuem formando uma "cauda" que aponta para um determinado lado da distribuição, conforme observado na Figura 4.9 nos gráficos *Skewed Left* e *Skewed Right*. Assim, de acordo com Doane & Seward [2011], a distribuição apresenta uma cauda que aponta para a direita ou esquerda, e, no caso da assimetria à direita, há um número pequeno de ocorrências de valores altos e um número grande de ocorrências de valores baixos. Esses casos não são "cauda-pesadas" por definição, mas o comportamento é semelhante em termos da significância da média na amostra. Nesses casos, apesar de os dados não se distribuírem como uma "cauda-pesada", optou-se por derivar os valores referências por meio das três faixas de valores, melhor traduzindo a forma como os dados se distribuem.

[7] A ferramenta R realiza a identificação do valor por meio da função *quantile*. Por exemplo, *quantile(values, c(.70))* retorna o 70% percentil do conjunto de dados *values*.

4.2 Avaliação dos Valores Referência

Os valores referência podem auxiliar especialistas a utilizarem métricas em suas tarefas, principalmente como uma forma de quantificar os atributos de qualidade interna do software. Os valores referência identificados nesta pesquisa, por meio da análise da distribuição de dados de um grande conjunto de softwares, são avaliados por meio de um experimento e três estudos de casos.

No catálogo proposto, além de valores referência para métricas de métodos e classes, são identificados valores referência para métricas de pacotes. Essas métricas avaliam a qualidade da estruturação do projeto, que, no processo de evoluação do software, sofre de um processo natural de degradação de sua qualidade [Lehman, 1978]. Reestrutar o projeto é uma forma de re-adequação da arquitetura, tornando as tarefas de manutenção mais fáceis por, pelo menos, mais um período de tempo [Anquetil & Laval, 2011]. Para avaliar esses valores referência, conduziu-se um experimento com o objetivo de verificar, em processos reais de reestruturação de pacotes orientados por objetos, se os valores referência para as métricas de pacotes são capazes de indicar quantitativamente a melhoria objetivada na reestruração. Esse experimento está descrito no Capítulo 7.

Ferreira et al. [2012] avaliaram os valores referência para métricas de classe por meio de estudos de casos, considerando que a aplicação desses valores pode resultar em duas situações possíveis: o valor referência avalia a classe corretamente ou não. A avaliação correta ocorre quando a faixa em que a métrica cai reflete a situação real da classe. Nos estudos de caso, consideram-se duas situações em que não há a avaliação correta do valor referência: casos *falso-positivos*, quando o valor referência avalia classe como ruim e a inspeção qualitativa não identifica problemas estruturais e casos *falso-negativos*, quando o valor não classifica a classe como ruim, mas a inspeção qualitativa identifica problemas em sua estrutura.

No presente trabalho, para a avaliação dos valores referência são conduzidos três estudos, discutidos a seguir.

4.2.1 Estudo de Caso 1

O Estudo de Caso 1 tem por objetivo avaliar um software proprietário com problemas de qualidade interna, com o propósito de verificar a capacidade dos valores referência propostos em indicar esse panorama. Para isto, investiga-se se a aplicação dos valores referência identificam métodos e classes com problemas estruturais quando realmente eles possuem tais problemas, evitando assim um dos possíveis erros quando da

utilização dos valores referência, que são avaliações *falso-negativas*. Em um software qualitativamente definido como ruim, espera-se que métodos e classes não sejam bem avaliados, refletindo o panorama de baixa qualidade. Esse estudo é divido em 3 partes, buscando avaliar, respectivamente, os valores referência propostos para as métricas de métodos, de classes e a aplicabilidade destes na identificação de *bad smells* bem conhecidos. A motivação para a condução deste estudo de caso é estender a avaliação dos valores referência para softwares proprietários e fora do *dataset* utilizado no processo de derivação. Segundo Spinellis [2008], os processos de desenvolvimento de softwares de código aberto e proprietários variam de forma drástica. Spinellis [2008] coletou métricas de software para avaliar atributos de qualidade interna de quatro grandes sistemas operacionais: FreeBSD, Linux, OpenSolaris e o Windows Research Kernel (WRK), com o objetivo de realizar uma comparação quantitativa de atributos de qualidade de código em softwares abertos e softwares proprietários. A contribuição principal desse trabalho foi a constatação de que não há diferenças significativas de qualidade de código entre os grandes sistemas estudados, apesar das características variáveis dos processos de desenvolvimento utilizados. Dada a constatação do trabalho de Spinellis [2008], bem como o conceito da programação orientada por objetos, espera-se que, apesar das diferenças significativas entre os processos de desenvolvimento de softwares abertos e proprietários, os valores referência derivados por meio do Qualitas.*class* Corpus sejam representativos também para softwares proprietários. Esse estudo de caso será tratado em detalhes no Capítulo 8.

4.2.2 Estudo de Caso 2

O Estudo de Caso 2 avalia uma amostra de métodos e classes do Qualitas.class *Corpus* classificados como *Ruim/Raro*, com o objetivo principal de verificar como os valores referência derivados geraram casos *falso-positivos*, ou se nem mesmo chegaram a gerar. Muitos casos *falso-positivos* na amostra verificada é um indício de que o valor referência não obteve sucesso em diferenciar o que é *Bom/Frequente* ou *Regular/Ocasional* do que é *Ruim/Raro*, indicando que o valor de fronteira entre as faixas *Regular/Ocasional* e *Ruim/Raro* pode estar baixo. Para a realização desse estudo, será necessário chegar a um número plausível de métodos e classes para a avaliação qualitativa, haja vista que a inspeção é essencialmente manual.

4.2.3 Estudo de Caso 3

No Estudo de Caso 3 é avaliado o software JHotDraw[8]. Esse software é considerado como um software de boa qualidade estrutural, maduro e que aplica extensivamente padrões de projetos. JHotDraw foi usado no trabalho de Ferreira et al. [2012] para avaliar a hipótese estudada pelos autores, e que também é o objetivo deste Estudo de Caso 3, de que a aplicação dos valores referência na avaliação dos métodos e classes desse software irá mostrar um panorama de qualidade. A confirmação da hipótese é sugestiva de que a aplicação dos valores referência possui baixo risco de resultados *falso-positivos*, ou seja, quando um método ou classe é mal avaliada pelo valor referência estabelecido sem que existam problemas estruturais, de fato. Esse resultado também corrobora a confiabilidade das avaliações positivas, sugerindo um baixo risco de casos *falso-negativos*.

Nos Estudos de Casos 2 e 3 serão avaliadas as duas situações em que não há avaliação correta do valor referência derivado: casos *falso-positivos* e *falso-negativos*. Espera-se que essa avaliação qualitativa seja capaz de sugerir a eficácia dos valores referência derivados. Esses estudos de casos, por estarem relacionados com a aplicação dos valores referência em softwares do próprio *dataset*, o Qualitas.*class* Corpus, serão descritos em detalhes no Capítulo 9.

4.3 Conclusão

No Capítulo 4 foi apresentado o método utilizado neste trabalho para extração de valores referência para métricas de softwares orientados por objetos. No capítulo seguinte será apresentada a aplicação deste método para a extração de valores referência de 18 métricas, culminando com a proposição de um catálogo de valores referência para métricas de software orientados por objetos.

[8] `http://www.jhotdraw.org`

Capítulo 5

Catálogo de Valores Referência

Este capítulo apresenta os resultados dos valores referência derivados a partir da pesquisa realizada. Seção 5.1 descreve as distribuições sugeridas pela ferramenta *EasyFit*, como melhor encaixe para as métricas estudadas e apresenta os conceitos fundamentais para o entendimento do método de derivação. Seção 5.2 descreve, para cada métrica, os resultados obtidos a partir do conjunto de dados. Seção 5.3 exibe o catálogo de valores para métricas de softwares orientados por objetos. Seção 5.4 apresenta discussões acerca da aplicação do método, buscando avançar em seu entendimento e interpretação de seus resultados.

5.1 Distribuições de Probabilidade

Uma distribuição de probabilidade modela a chance de uma variável assumir determinado valor ao longo do espaço de valores. Ela é uma função cujo domínio são os valores da variável, no caso, as métricas estudadas, e cuja imagem são as probabilidades de aquele valor ocorrer.

Um problema importante na Estatística é obter informação sobre a distribuição seguida pelo conjunto de dados analisado. Para esse propósito, EasyFit [2013], por meio do teste de aderência de *Kolmogorov-Smirnov*, realiza a seleção da distribuição apropriada aos dados. Selecionar a distribuição apropriada aos dados é o procedimento de escolher uma distribuição que tenha o melhor ajuste para um conjunto de dados. O objetivo de selecioná-la, nesse contexto, é estabelecer a distribuição apropriada a cada métrica de software estudada.

Esta seção descreve o conjunto de distribuições retornadas e analisa suas principais características de interesse para a derivação dos valores referência. Essa descrição

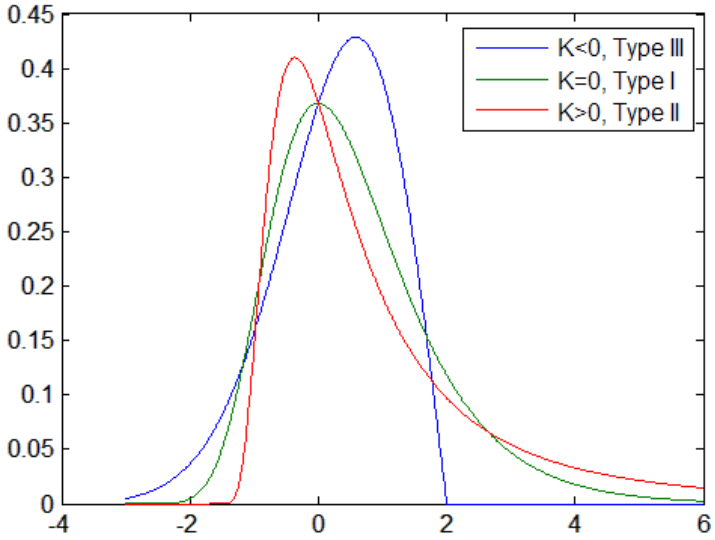


Figura 5.1: Distribuição *Generalized Extreme Value*. Fonte: Matlab [2014]

se baseia na documentação da própria ferramenta EasyFit e na documentação de outras duas ferramentas estatísticas: Matlab [2014] e VoseSoftware [2014].

5.1.1 *Generalized Extreme Value*

A função densidade de probabilidade da distribuição *Generalized Extreme Value* possui os seguintes parâmetros: parâmetro de localização, μ; parâmetro de escala, σ, e o parâmetro de forma, k. Os parâmetros μ e σ não influenciam na formação da cauda. O parâmetro k indica o comportamento da cauda na distribuição. Existem 3 possibilidades: $k < 0$, $k = 0$ e $k > 0$. Distribuições cujas caudas diminuem exponencialmente, como a *Normal* levam ao Tipo 1 ($k = 0$); distribuições cujas caudas diminuem como uma função polinomial levam ao Tipo 2 ($k > 0$); e distribuições cujas caudas existem e são finitas levam ao Tipo 3 ($k < 0$). A Figura 5.1 ilustra o comportamento do parâmetro de forma, k. Percebe-se que a distribuição *Generalized Extreme Value* possui assimetria à direita, ou assimetria positiva.

5.1.2 *Log-Logistic*

De acordo com Beirlant et al. [2001], a distribuição *Log-Logistic* é uma distribuição de característica cauda-pesada, sendo similar a distribuição *Log-Normal* (também cauda-pesada), porém com caudas ainda "mais" pesadas. A função densidade de probabilidade da distribuição *Log-Logistic* possui os seguintes parâmetros: parâmetro de forma, α e parâmetro de escala, β. O aumento do parâmetro α em relação a β aumenta a altura da curva, enquanto que o inverso causa a diminuição da altura da curva, achatando-a.

A Figura 5.2 ilustra o comportamento da cauda em relação aos parâmetros, por meio da função $LogLogistic(\alpha, \beta)$.

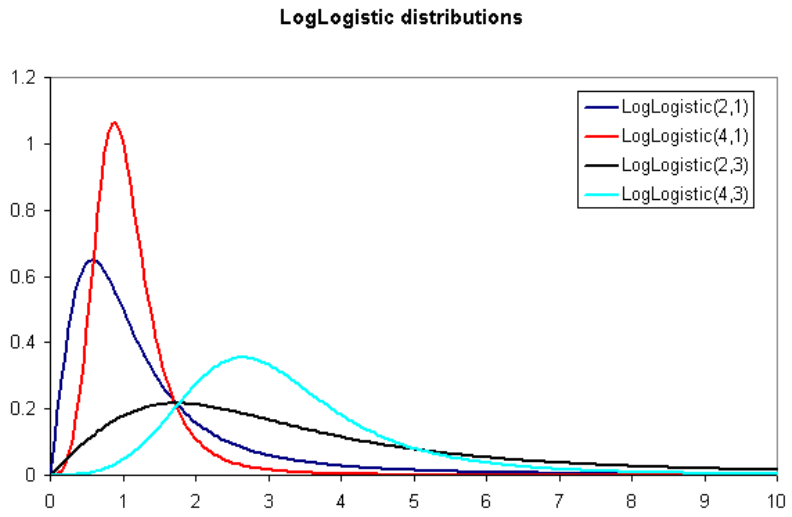


Figura 5.2: Distribuição *Log-Logistic*. Fonte: VoseSoftware [2014]

5.1.3 *Pareto 2*

Segundo Van & Vose [2008], *Pareto 2* é uma distribuição de cauda-pesada, que, essencialmente, é a distribuição de Pareto padrão, com um deslocamento ao longo do eixo x para ser iniciada em $x = 0$. A função densidade de probabilidade da distribuição *Pareto 2* possui os seguintes parâmetros: parâmetro de forma, α e o parâmetro de escala, β. O aumento do parâmetro β em relação a α tem o efeito de diminuir a altura da curva, achatando-a. Já o aumento do parâmetro α em relação a β tem efeito contrário, aumentando a altura da curva. Logo, quanto maior for $|\alpha - \beta|$, maior será a altura curva.

A Figura 5.3 ilustra o comportamento da cauda em relação aos parâmetros, por meio da função $Pareto2(\beta, \alpha)$. Percebe-se que, tanto a linha preta como a linha vermelha possuem $\alpha = 2$. Aumentando o pârametro $\beta = 0,5$ na linha preta para $\beta = 4$ na linha vermelha causa uma diminuição na altura da curva. Observa-se também que, para a linha azul, tem-se $|\alpha - \beta| = 1$, para a linha vermelha tem-se $|\alpha - \beta| = 2$ e para a linha preta têm-se $|\alpha - \beta| = 4$. Percebe-se que, quanto maior o valor de $|\alpha - \beta|$, mais alta é a curva, havendo uma potencialização da característica de cauda-pesada.

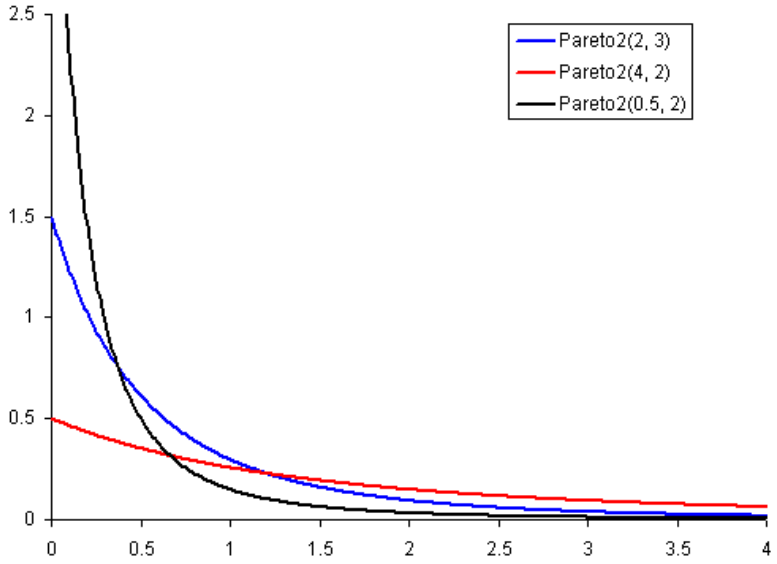


Figura 5.3: Distribuição *Pareto 2*. Fonte: VoseSoftware [2014]

5.1.4 *Weibull*

A distribuição *Weibull* possui os seguintes parâmetros: parâmetro de forma, α, e o parâmetro de escala, β, sendo considerada uma distribuição cauda-pesada quando $\alpha < 1$ [Van & Vose, 2008; Foss et al., 2009; EasyFit, 2013]. O aumento do parâmetro β em relação a α tem o efeito de diminuir a altura da curva, achatando-a. O aumento do parâmetro α em relação a β tem efeito contrário, aumentando a altura da curva. Assumindo $\beta = 1$, as possibilidades são: se $\alpha < 1$, *Weibull* é cauda-pesada; se $\alpha = 1$, *Weibull* é equivalente a distribuição exponencial; se $1 < \alpha < 3,25$, *Weibull* apresenta assimetria à direita, sendo que, quanto menor o valor de α, maior será a assimetria; se $\alpha = 3,25$, *Weibull* é similar a distribuição *Normal* e se $\alpha > 3,25$, *Weibull* apresenta uma curva aproximadamente simétrica, porém mais pontiaguda do que a distribuição *Normal*. A Figura 5.4 ilustra o comportamento da cauda em relação aos parâmetros, por meio da função $Weibull(\alpha, \beta)$.

Percebe-se que quando $\alpha \geq 1$, nos casos em que o valor do parâmetro fica próximo de 1, a distribuição apresenta considerável assimetria à direita, e torna-se mais "ajustada" com a aproximação de 3,25. Nesses casos, de acordo com Doane & Seward [2011], a distribuição apresenta uma cauda longa que aponta para a direita, havendo um número pequeno de ocorrências de valores altos e um número grande de ocorrências de valores baixos. Esses casos não são "cauda-pesadas" por definição, mas o comportamento é semelhante. Assim, neste trabalho, quando os valores de uma métrica são modelados dessa forma, optou-se por derivar os valores referências por meio de faixas

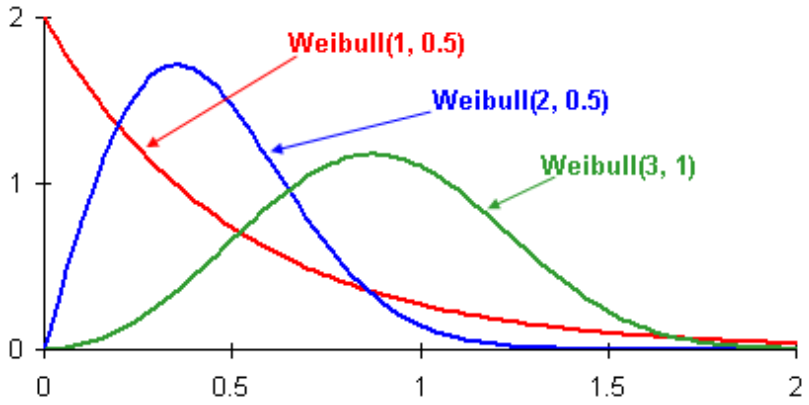


Figura 5.4: Distribuição *Weibull*. Fonte: VoseSoftware [2014]

de valores.

5.1.5 *Generalized Pareto*

A função densidade de probabilidade da distribuição *Generalized Pareto* possui os seguintes parâmetros: parâmetro de localização, μ; parâmetro de escala, σ, e parâmetro de forma, k. Os parâmetros μ e σ não influenciam na formação da cauda. O parâmetro k indica o comportamento da cauda na distribuição.

Existem 3 possibilidades: $k < 0$, $k = 0$ e $k > 0$. Tal como a distribuição *Generalized Extreme Value*, relatada na Seção 5.1.1, distribuições cujas caudas diminuem exponencialmente, como a *Normal*, levam ao Tipo 1 ($k = 0$); distribuições cujas caudas diminuem como uma função polinomial, levam ao Tipo 2 ($k > 0$) e distribuições cujas caudas existem e são finitas, levam ao Tipo 3 ($k < 0$). Como pode ser observado na Figura 5.5, as curvas são assimétricas, possuindo uma cauda-longa que aponta para a direita. Quando $k < 0$, diminuir o parâmetro k significa aumentar a largura da cauda até o limite de -1. Quando $k < -1$, a distribuição apresenta assimetria à esquerda. Diminuir o parâmetro μ tem o efeito de aumentar a altura da cauda, achatando-a, e vise-versa. Diminuir o parâmetro σ tem o efeito de deslocar a curva para a esquerda, e vice-versa.

5.1.6 *Power Function*

A distribuição *Power Function* possui os seguintes parâmetros: parâmetro de forma, α e parâmetros de fronteira, a e b, com $a < b$ [Foss et al., 2009; EasyFit, 2013]. Caracteriza-se como uma distribuição cauda-pesada quando $\alpha < 1$. O aumento do

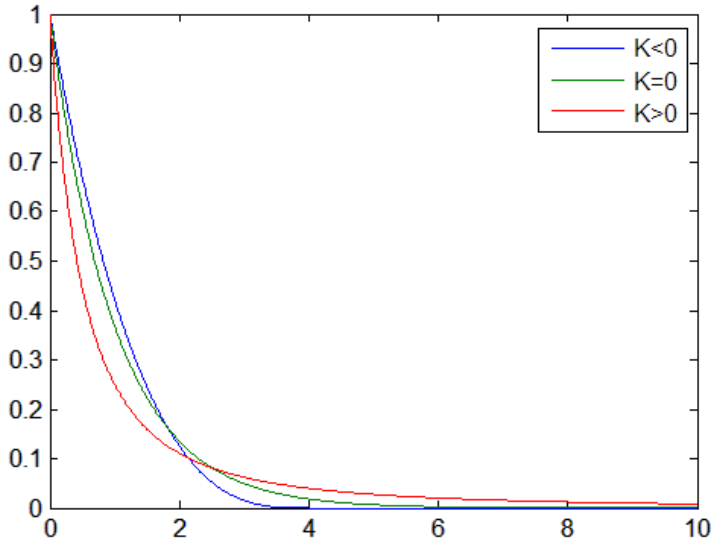


Figura 5.5: Distribuição *Generalized Pareto*. Fonte: Matlab [2014]

parâmetro a em relação a b tem o efeito de aumentar a altura da curva, enquanto que o aumento de b em relação a a tem o efeito de achatá-la.

5.1.7 *Beta*

A distribuição *Beta* possui os seguintes parâmetros: dois parâmetros de forma, α_1 e α_2, e parâmetros de fronteira, a e b, com $a < b$ [EasyFit, 2013]. De acordo com Van & Vose [2008], ela pode tomar uma grande quantidade de formas, como mostra a Figura 5.6. Percebe-se que nos casos em que $\alpha_1, \alpha_2 > 1$, quando $\alpha_1 < \alpha_2$, a distribuição possui assimetria à direita, sendo que quanto maior a distância entre os dois parâmetros, maior é a assimetria, ilustrado no primeiro gráfico da Figura 5.6. O segundo gráfico mostra os casos em que a distribuição apresenta retas. O terceiro gráfico ilustra os casos que apresentam $\alpha_1, \alpha_2 < 1$. Percebe-se que, nesses casos, quando a distribuição possui $\alpha_1 < 1$ e $\alpha_2 < 1$, ela apresenta um número pequeno de ocorrências entre os extremos das distribuições e um número grande de ocorrência de valores baixos e altos, respectivamente. Quando $\alpha_1 < 1$ e $\alpha_2 > 1$, a distribuição possui um número grande de ocorrências de valores baixos, constituindo a cauda-pesada. Além disso, observa-se que o aumento do parâmetro a em relação a b tem o efeito de aumentar a altura da curva, enquanto que o aumento de b em relação a a tem o efeito de achatá-la.

5.1.8 *Gumbel Max*

A distribuição *Gumbel Max* possui os seguintes parâmetros: parâmetro de escala, σ, e parâmetro de localização, μ [EasyFit, 2013]. A forma da distribuição possui assimetria

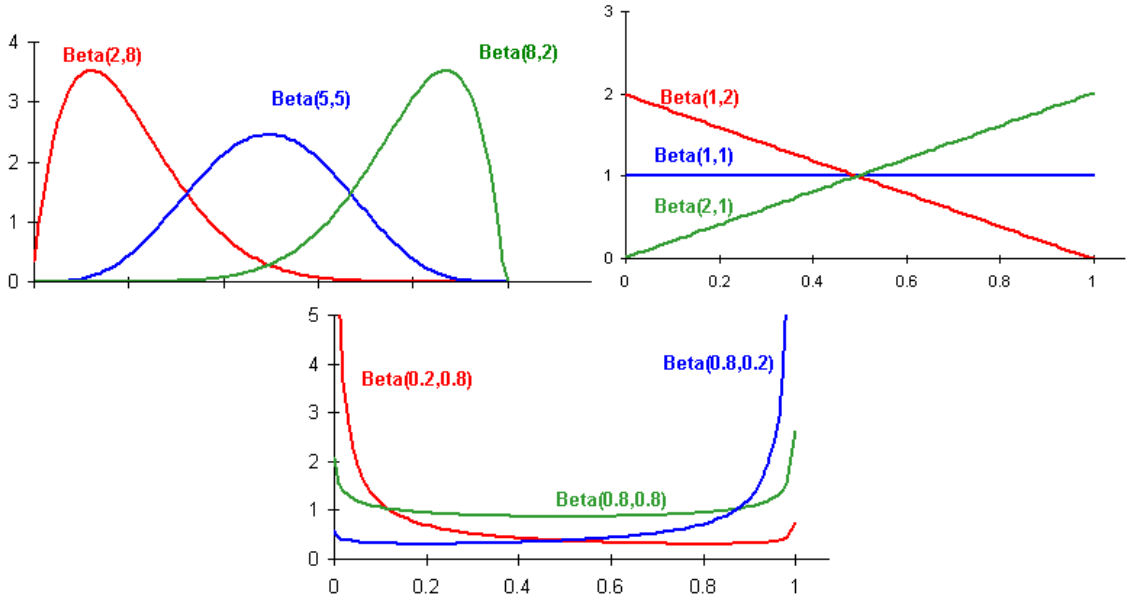


Figura 5.6: Distribuição *Beta*. Fonte: VoseSoftware [2014]

à direita, apresentando uma cauda longa que aponta para a direita, havendo um número pequeno de ocorrências de valores altos e um número grande de ocorrências de valores baixos. Conforme μ diminui, o eixo x desloca-se para a esquerda, e vice-versa. Conforme σ diminui, a altura da curva aumenta, e o seu aumento tem o efeito de diminuir a altura da curva, achatando-a. A Figura 5.7 ilustra a forma assimétrica da curva.

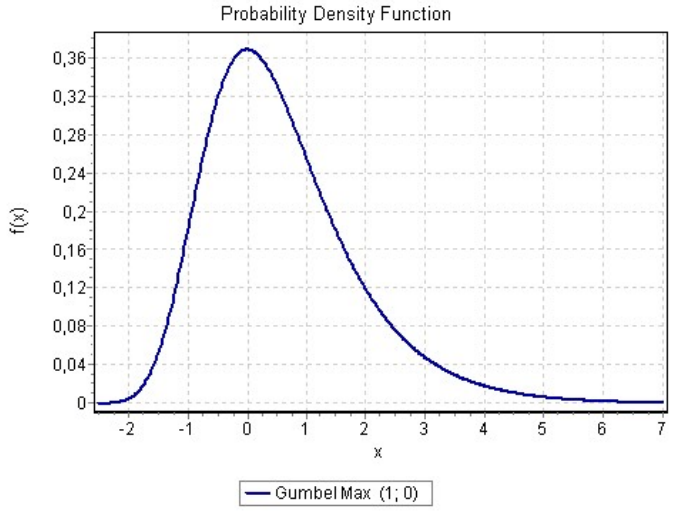


Figura 5.7: Distribuição *Gumbel Max*. Fonte: EasyFit [2013]

5.1.9 *Chi-Squared*

A distribuição *Chi-Squared* possui os seguintes parâmetros: parâmetro de grau de liberdade, ν (inteiro positivo), e parâmetro de localização, γ, sendo uma distribuição de cauda-pesada [EasyFit, 2013]. A Figura 5.8 ilustra o comportamento do parâmetro ν. Apesar de não estar caracterizado na figura, quando têm-se $\nu = 1$, os valores baixos apresentam o máximo de frequência possível para a distribuição. Quando o parâmetro ν aumenta, a cauda se achata. A distribuição apresenta assimetria à direita, sendo ela máxima e de característica cauda-pesada quando $\nu = 1$. O aumento do parâmetro γ causa o deslocamento positivo no eixo x, enquanto que a diminuição causa o deslocamento negativo.

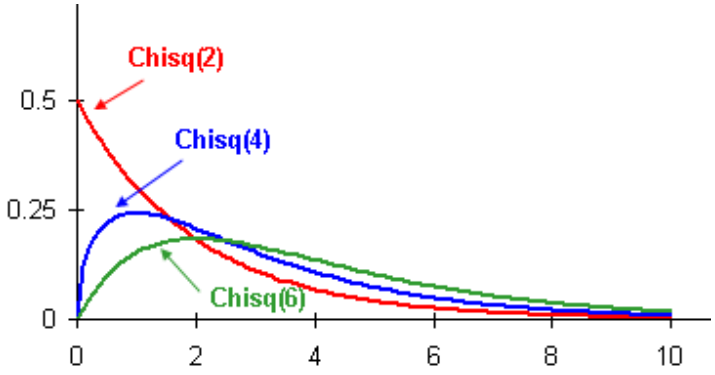


Figura 5.8: Distribuição *Chi-Squared*. Fonte: VoseSoftware [2014]

5.2 Valores Referência

Esta seção descreve, para cada métrica, os resultados obtidos a partir do conjunto de dados. Os gráficos dos valores coletados para as métricas descritas nas Seções 5.2.1 a 5.2.14 sugerem uma distribuição de cauda-pesada. A fim de tornar o texto dessas seções mais conciso, optamos por descrever na íntegra a métrica Acoplamento Aferente (CA), mostrando todo o processo usado na derivação dos valores referência sugeridos para ela na Seção 5.2.1. Para outras 13 métricas (Seções 5.2.2 a 5.2.14), apresentaremos somente as diferenças. A descrição completa dessas métricas pode ser vista no Apêndice A deste texto, nas Seções A.1.1 a A.1.13. As métricas DIT, LCOM, NBD e RMD possuem características diferentes e, portanto, o processo de derivação dos valores referência sugeridos para elas será descrito na íntegra, respectivamente nas Seções 5.2.15, 5.2.16, 5.2.17 e 5.2.18. A distribuição de melhor ajuste ao conjunto de valores das métricas estudadas foi retornado pela ferramenta *EasyFit*, conforme descrito na Seção 4.1.2.2.

5.2.1 Acoplamento Aferente (CA)

O Gráfico de Dispersão dos valores coletados para a métrica Acoplamento Aferente (CA), ilustrado na Figura 5.9a, sugere uma distribuição de cauda-pesada. A Figura 5.9b ressalta ainda mais essa característica. É possível observar que a aproximação dos 100% de frequência acumulada ao longo do eixo x (valores das métricas) ocorre de uma forma drasticamente rápida, ou seja, há uma aproximação quase que instantânea dos 100% dos valores. Isso significa que há um grande número de pacotes com poucas classes externas que dependem de suas classes e um pequeno número de pacotes com muitas classes externas que dependem de suas classes.

Como mostrado nas Figuras 5.9c e 5.9d, o conjunto de valores dessa métrica possui melhor ajuste à distribuição *Generalized Extreme Value*, com parâmetros $k = 0,797$, $\sigma = 4,303$ e $\mu = 1,775$, conforme retornado pela ferramenta *EasyFit*. A Figura 5.10 mostra a Função Densidade de Probabilidade com os parâmetros retornados, sem apresentar ajuste aos dados. Percebe-se, na visualização desse gráfico, a característica de cauda-pesada da distribuição com esses parâmetros.

A Figura 5.9e mostra o Histograma de Probabilidade. A título de visualização, foi utilizado o 99° percentil dos dados, que determina os 99% menores dados do conjunto. Percebe-se que a segunda barra, que representa os valores de CA entre 20 e 40, constitui menos que 10% dos dados. Novamente, fica ressaltada a característica da distribuição dessa métrica, na qual valores baixos são muito frequentes, caracterizando a situação típica dos pacotes nos projetos de software. Valores intermediários tem representatividade, mas, mesmo assim, são baixos em relação a amplitude dos valores coletados. Valores altos são extremamente raros e caracterizam situações fora do comum da maioria dos pacotes. A Figura 5.9f mostra esses dados em escala logarítmica. Nesse gráfico, percebe-se uma reta inclinada à esquerda, característica de uma lei de potência. Isso reforça ainda mais as características já citadas, em que a grande maioria dos pacotes possuem poucas classes externas que dependem de suas classes.

Foram identificados os valores que representam 70° e 90° percentil dos dados, que correspondem aos valores 7 e 39. Os valores 70% e 90% foram identificados por meio de uma análise visual no Gráfico de Frequência Relativa Acumulada da Figura 5.9b. Os pontos marcados com a cor verde / quadrado e vermelha / losango representam as regiões identificadas nessa análise. Além disso, pesa na escolha desses percentis os conceitos das faixas *Bom/Frequente*, *Regular/Ocasional* e *Ruim/Raro*, que devem caracterizar, respectivamente, valores muito frequentes, que ocorrem ocasionalmente e raros. Dessa forma, baseado na análise visual, no conceito estabelecido para as faixas, e inspirado no trabalho de Alves et al. [2010], que utiliza percentis para definir

estatisticamente perfis de qualidade em métricas de acordo com quatro categorias de classificação de risco: risco baixo (entre 0 e 70%), risco moderado (entre 70% e 80%), alto risco (entre 70% e 90%) e altíssimo risco (maior que 90%), tentou-se, neste trabalho, aplicar os valores 70° e 90° na maioria das métricas para separar as faixas com os valores sugeridos. Existem variações que vão de acordo com a característica da curva da distribuição, quando são mais altas ou achatadas, ou dependendo de um menor ou maior tempo necessário para atingir um maior acúmulo de frequência, situações nas quais os valores 70° e 90° não teriam significância. Nesses casos, foi usada majoritariamente a análise visual dos pontos que são capazes de separar as faixas de valores.

Como 70° e 90° percentil dos dados correspondem aos valores 7 e 39, tem-se que 70% dos pacotes possuem $CA \leq 7$ e 90% dos pacotes possuem $CA \leq 39$. Esses valores podem ser obtidos de forma automatizada com a utilização de ferramentas estatísticas como R[1]. Isso é importante pois torna o estudo reprodutível. Conforme observado na Seção 4.1.2.4, esses valores permitem separar a métrica CA em 3 grupos: *Ruim/Raro*, *Regular/Ocasional* e *Bom/Frequente*. A Tabela 5.1 sumariza os valores referência derivados para a métrica Acoplamento Aferente (CA).

Tabela 5.1: Valores referência para CA

Bom/Frequente	**Regular/Ocasional**	**Ruim/Raro**
$CA \leq 7$	$7 < CA \leq 39$	$CA > 39$

Conforme relatado na Seção 2.3, a métrica CA mede o número de classes externas a um determinado pacote que dependem de suas classes internas. Quanto maior for o valor de CA, maior é a responsabilidade daquele pacote, sendo um indicativo de sua relevância dentro da arquitetura do software. Um pacote com muitas dependências externas torna-se um artefato de risco, haja vista que, quando for necessário realizar alguma alteração nele, isso pode impactar diretamente em muitas classes e indiretamente ter consequências na estabilidade de toda a aplicação. Pacotes com valores altos de CA devem ser cuidadosamente monitorados, pois qualquer mudança pode se tornar crítica ao software, devendo esses pacotes receberem um maior esforço de testes e monitoração. Dessa forma, os valores referência aqui sugeridos são úteis no sentido de serem capazes de dizer o que é um valor alto de CA, baseado nos padrões de qualidade dos softwares desenvolvidos.

Conforme relatado por Lanza et al. [2007], as responsabilidades dentro de um software orientado por objetos devem estar uniformemente distribuídas nos artefatos

[1] http://www.r-project.org/

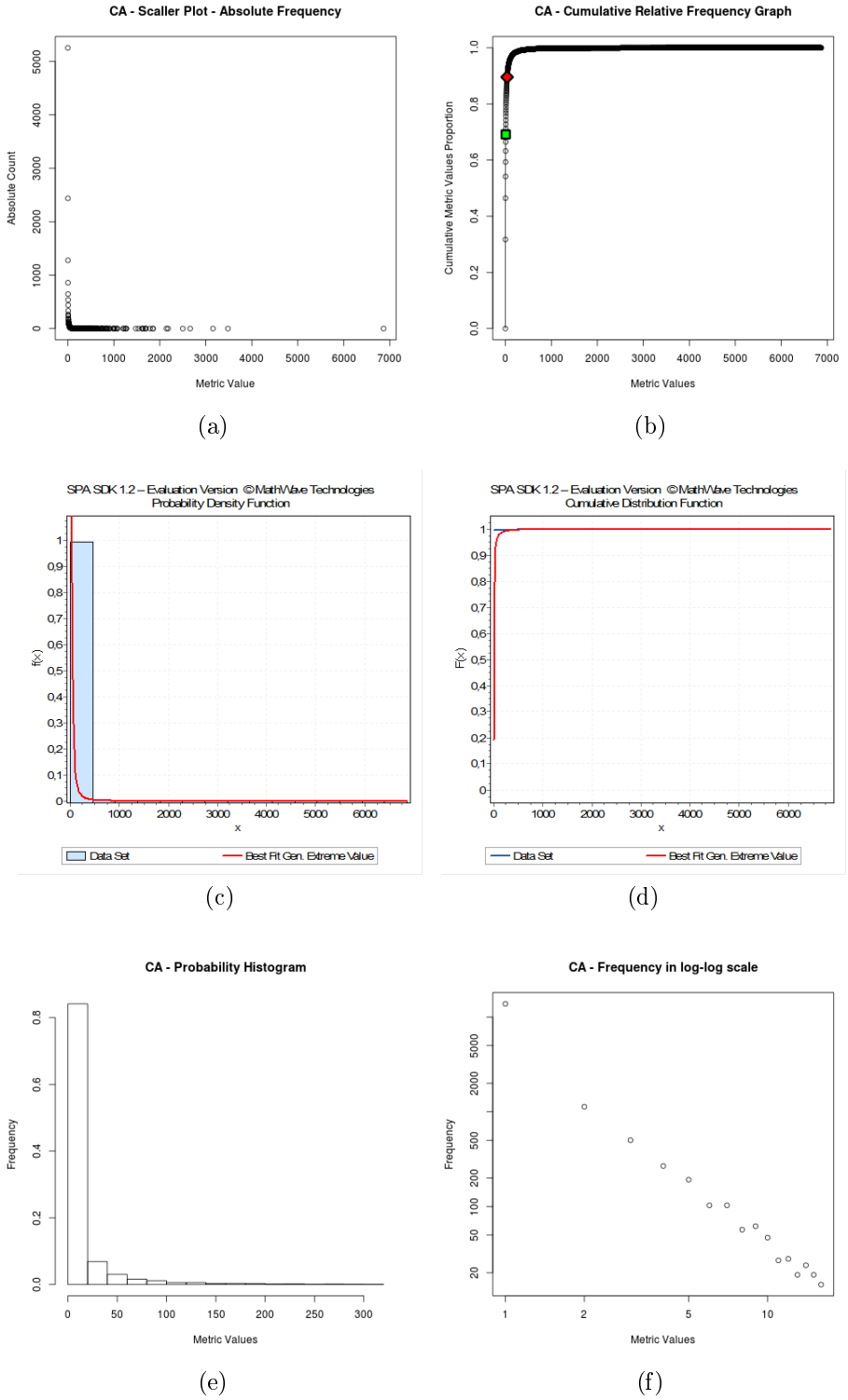


Figura 5.9: Acoplamento Aferente: (a) Gráfico de Dispersão (b) Gráfico de Frequência Relativa Acumulada (c) Função de Densidade de Probabilidade ajustada à Distribuição Generalized Extreme Value (d) Gráfico de Frequência Relativa Acumulada ajustado à Distribuição Generalized Extreme Value (e) Histograma de Probabilidade (f) Histograma em escala *log-log*.

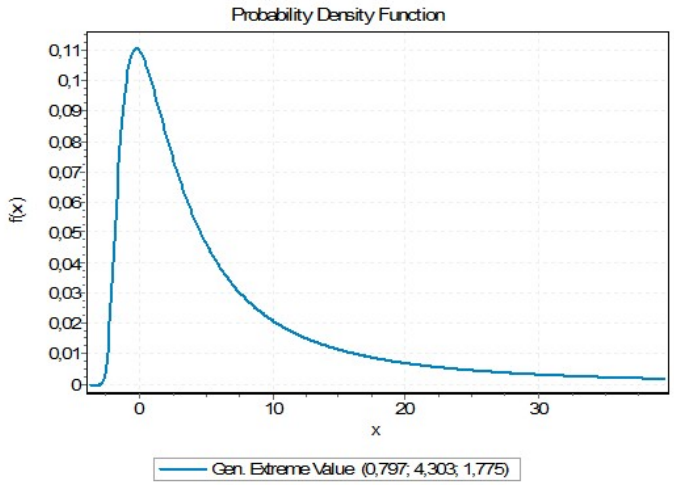


Figura 5.10: CA - Função de Densidade de Probabilidade com os parâmetros retornados pela ferramenta *EasyFit*

de software. Por isso, saber o que é um valor alto de CA permite direcionar esforços de análise para avaliar uma possível redistribuição das responsabilidades de um pacote demasiadamente influente em um conjunto de pacotes mais coesos, isso é, que agrupem melhor classes que tenham objetivos comuns e tendem a ser reutilizadas em conjunto.

5.2.2 Acoplamento Eferente (CE)

O conjunto de valores para a métrica Acoplamento Eferente (CE) possui melhor ajuste à distribuição *Generalized Extreme Value*, com parâmetros $k = 0,527$, $\sigma = 2,834$ e $\mu = 2,397$. Foram identificados os valores que representam 70° e 90° percentil dos dados, que correspondem aos valores 6 e 16, sumarizados na Tabela 5.2.

Tabela 5.2: Valores referência para CE

Bom/Frequente	**Regular/Ocasional**	**Ruim/Raro**
$CE \leq 6$	$6 < CE \leq 16$	$CE > 16$

A métrica CE mede o número de classes de um determinado pacote que dependem de classes externas a esse pacote. De acordo com Metrics [2014], quanto maior o valor de CE, mais o pacote depende de detalhes de implementação de outras classes, tornando-o um artefato mais instável, dado o alto grau de classes dependentes. Manter o valor de CE mínimo significa obter um pacote com maior autossuficiência, provendo serviços de forma mais estável, com dependências externas limitadas ao objetivo de servir a um propósito bem direcionado.

Os valores referência derivados sugerem que a maioria dos pacotes em softwares orientados por objetos tem sido desenvolvidos com um grau de qualidade relacionado a essa métrica, em que não se deve ter mais de 6 classes dependentes. Ocasionalmente se tem até 16 classes dependentes e raramente se tem mais que 16 classes dependentes. Com esses valores derivados, é possível dizer o que é um CE alto, regular e bom dentro dos pacotes de software, apoiando a decisão em relação a aspectos de qualidade de software no que tange a essa métrica. Uma possível decisão acerca de um pacote de software com alto valor de CE (*Ruim/Raro*) seria dividi-lo em pacotes menores e possivelmente mais coesos.

5.2.3 Linhas de Código por Método (MLOC)

O conjunto de valores para a métrica Linhas de Código por Método (MLOC) possui melhor ajuste à distribuição *Pareto 2*, com parâmetros $\alpha = 1,226$ e $\beta = 3,051$. Foram identificados os valores que representam o 80° e 95° percentil dos dados, correspondendo aos valores 10 e 30, apresentados na Tabela 5.3.

Tabela 5.3: Valores referência para MLOC

Bom/Frequente	**Regular/Ocasional**	**Ruim/Raro**
$MLOC \leq 10$	$10 < MLOC \leq 30$	$MLOC > 30$

Segundo Fowler [1999], desde os primórdios da programação, percebeu-se que, quanto maior é um procedimento, mais difícil é entendê-lo. O autor sugere que grande parte dos problemas em softwares orientados por objetos tem origem em métodos longos, que são problemáticos por conter muita complexidade e informação. Métodos menores possuem maior valor para os softwares, pois são mais fáceis de entender, reutilizar e sobrescrever, tornando a manutenção e evolução do sistema uma tarefa menos complexa e, consequentemente, mais barata. Logo, é consenso que métodos pequenos tem impacto direto na qualidade de software. Sommerville [2010] relata que a extensão de código é uma das métricas mais confiáveis de previsão de propensão a erros. Baseado na análise estatística realizada, sugere-se que métodos não devam ter mais do que 10 linhas de código. Isso está de acordo com as melhores práticas estabelecidas pela comunidade de desenvolvimento de software, que são baseadas em um senso comum de entendimento qualitativo. Comprova-se aqui, com uma pesquisa empírica, que a grande maioria dos métodos segue esse padrão de qualidade. Acredita-se que os valores aqui sugeridos podem guiar de forma quantitativa as boas práticas de desenvolvimento de software.

5.2.4 Número de Classes por Pacote (NOC)

O conjunto de valores para a métrica Número de Classes por Pacote (NOC) possui melhor ajuste à distribuição *Log-Logistic*, com parâmetros $\alpha = 1,452$ e $\beta = 5,520$. Foram identificados os valores que representam o 70° e 90° percentil dos dados, que correspondem aos valores 11 e 28, mostrados na Tabela 5.4.

Tabela 5.4: Valores referência para NOC

Bom/Frequente	**Regular/Ocasional**	**Ruim/Raro**
$NOC \leq 11$	$11 < NOC \leq 28$	$NOC > 28$

Conforme relatado em Oracle [2014c], um pacote é um agrupamento de classes e interfaces relacionadas a um mesmo domínio ou propósito, provendo proteção de acesso e facilidade de localização. Quanto mais classes forem adicionadas a um determinado pacote, menos inter-relacionado o agrupamento de classes e interfaces tende a ficar, sugerindo uma possível divisão em pacotes menores, de forma a criar novos agrupamentos que traduzam um domínio mais bem definido. Isso possibilita encontrar mais facilmente classes relacionadas, permitindo entender aquele domínio mais rapidamente, o que torna a manutenção mais fácil. Os valores referência aqui sugeridos são capazes de identificar pacotes com uma quantidade de classes que foge do padrão da maioria dos pacotes desenvolvidos nos softwares orientados por objetos, permitindo avaliar possíveis adequações a serem realizadas dentro do projeto.

5.2.5 Número de Atributos (NOF)

O conjunto de valores para a métrica Número de Atributos (NOF) possui melhor ajuste à distribuição *Beta*, com parâmetros $\alpha_1 = 0,059$, $\alpha_2 = 64,580$, $a = 0,000$ e $b = 2026,446$. Foram identificados os valores que representam o 80° e 95° percentil dos dados, que correspondem aos valores 3 e 8, sumarizados na Tabela 5.5.

Tabela 5.5: Valores referência para NOF

Bom/Frequente	**Regular/Ocasional**	**Ruim/Raro**
$NOF \leq 3$	$3 < NOF \leq 8$	$NOF > 8$

De acordo com Fowler [1999], uma classe com muitos atributos é um indício de que sua modelagem pode conter algum problema, devendo assim ser reestruturada a partir do agrupamento de atributos relacionados em novos objetos. Essas classes tornam-se problemáticas por armazenarem muitos estados, carregando uma complexidade desnecessária, tornando-se difíceis de se manter e evoluir. Dessa forma, os valores

referência definidos podem ser utilizados para identificar classes candidatas a passar por um processo de refatoração, que pode elevar o nível de qualidade do software. De acordo com os valores referência aqui sugeridos, uma classe com mais de 8 atributos deve ser candidata à refatoração.

5.2.6 Número de Métodos (NOM)

O conjunto de valores para a métrica Número de Métodos (NOM) possui melhor ajuste à distribuição *Weibull*, com parâmetros $\alpha = 0,852$ e $\beta = 5,879$. Foram identificados os valores que representam o 70° e 90° percentil dos dados, que correspondem aos valores 6 e 14, apresentados na Tabela 5.6.

Tabela 5.6: Valores referência para NOM

Bom/Frequente	**Regular/Ocasional**	**Ruim/Raro**
$NOM \leq 6$	$6 < NOM \leq 14$	$NOM > 14$

De acordo com Lanza et al. [2007], em um bom projeto orientado por objetos, a inteligência do sistema deve ser uniformemente distribuída entre as classes. Uma classe com muitos métodos tende a ser mais complexa e assumir muitas responsabilidades, quebrando um dos principais princípios da orientação por objetos que sugere que uma classe deve assumir apenas uma responsabilidade. Nesse direcionamento, o valor referência derivado traduz que as classes são construídas, em sua maioria, com no máximo 6 métodos, sendo que classes com até 14 métodos ocorrem de forma ocasional. Classes com mais de 14 métodos são raras e devem ser candidatas à refatoração.

5.2.7 Número de Métodos Sobrescritos (NORM)

O conjunto de valores para a métrica Número de Métodos Sobrescritos (NORM) possui melhor ajuste à distribuição *Power Function*, com parâmetros $\alpha = 0,006$, $a = 0,000$ e $b = 235,200$. Foram identificados os valores que representam o 90° e 97° percentil dos dados, que correspondem aos valores 2 e 4, mostrados na Tabela 5.7.

Tabela 5.7: Valores referência para NORM

Bom/Frequente	**Regular/Ocasional**	**Ruim/Raro**
$NORM \leq 2$	$2 < NORM \leq 4$	$NORM > 4$

Segundo Sommerville [2010], um valor alto dessa métrica indica que a superclasse usada pode não ser uma classe mãe apropriada para a subclasse, sugerindo problemas

na hierarquia de herança do projeto. Logo, o valor referência derivado como *Bom/-Frequente* sugere que, tipicamente, as classes filhas sobrescrevam até dois métodos das classes mãe. Isso sugere que as classes filhas estão, de fato, reutilizando comportamentos das classes-mãe, sobrescrevendo uma quantidade reduzida de métodos.

5.2.8 Número de Filhas (NSC)

O conjunto de valores para a métrica Número de Filhas (NSC) possui melhor ajuste à distribuição *Beta*, com parâmetros $\alpha_1 = 0,001$, $\alpha_2 = 9,467$, $a = 0,000$ e $b = 25465,529$. Foram identificados os valores que representam o 85° e 95° percentil dos dados, que correspondem aos valores 1 e 3, sumarizados na Tabela 5.8.

Tabela 5.8: Valores referência para NSC

Bom/Frequente	**Regular/Ocasional**	**Ruim/Raro**
$NSC \leq 1$	$1 < NSC \leq 3$	$NSC > 3$

Chidamber & Kemerer [1994] sugerem que classes com um grande número de filhas são difíceis de modificar e irão exigir um maior esforço de testes no sistema, afinal, uma alteração na classe mãe poderá afetar todas suas classes filhas. Além disso, quanto maior for o número de filhas, maior é a probabilidade da abstração da classe mãe ser utilizada de forma imprópria e o projeto usar a herança de forma indevida. Dado o exposto, acredita-se que os valores referência sugeridos neste trabalho traduzam que uma classe mãe que possua 4 ou mais classes filhas pode representar um risco grande ao ser alterada pois pode impactar em um número considerável de outras classes. Além disso, é provável que em 4 ou mais classes filhas de uma mesma mãe haja comportamentos comuns a um sub-conjunto dessas classes, devendo-se criar um novo nível na hierarquia de forma a balancear a árvore de herança.

Os valores referência sugeridos estão de acordo com outros trabalhos, conforme observado na Seção 3.3. Os valores 2 e 3 foram derivados em outros estudos, resultado que está de acordo com a faixa *Regular/Ocasional* aqui estabelecida. Como nesse trabalho foram usadas três faixas de valores, são distinguidas também as faixas *Bom/Frequente* e *Regular/Ocasional*.

5.2.9 Número de Atributos Estáticos (NSF)

O conjunto de valores para a métrica Número de Atributos Estáticos (NSF) possui melhor ajuste à distribuição *Beta*, com parâmetros $\alpha_1 = 0,018$, $\alpha_2 = 18,284$, $a = 0,000$

e $b = 4130,615$. Foram identificados os valores que representam o 85° e 95° percentil dos dados, que correspondem aos valores 1 e 5, apresentados na Tabela 5.9.

Tabela 5.9: Valores referência para NSF

Bom/Frequente	**Regular/Ocasional**	**Ruim/Raro**
$NSF \leq 1$	$1 < NSF \leq 5$	$NSF > 5$

De acordo com Oracle [2014c], um atributo estático cria um campo que pertence à classe em vez de estar associado a uma instância daquela classe. Todas as instâncias da classe compartilham o atributo estático, que fica em um lugar fixo na memória. Qualquer alteração de valor nesses atributos refletirá em todas as instâncias da classe, afinal, o atributo é associado à classe e não à instância.

Segundo Bloch [2008], atributos estáticos possuem grande utilidade nos projetos de softwares orientados por objetos desenvolvidos na plataforma Java. Um exemplo é a implementação do padrão de projeto *Singleton*, que garante a existência de apenas uma instância de determinada classe, provendo acesso global a esse objeto. Atributos estáticos também podem ser expostos via modificadores *public static final*, assumindo que constantes formam uma parte coesa da abstração provida pela classe. No entanto, classes que utilizam excessivamente esse recurso tem adquirido uma má reputação pois evita que desenvolvedores pensem em termos de objetos.

Um exemplo de má utilização é a *constant interface*, relatada em Bloch [2008]. São interfaces compostas unicamente por atributos *static final*, cada um deles representando a exportação de uma constante. Classes implementam essas constantes apenas para evitar a necessidade de qualificar o nome da *interface*. Como esses atributos são utilizados na implementação das funcionalidades dessas classes, detalhes ficam expostos e a classe sofre risco de a ocultação da informação ficar prejudicada. Se a *interface* for alterada, o comportamento da classe herdeira também pode ser alterado, ou seja, não há um bom encapsulamento do projeto. Uma alternativa que traduz melhor a programação orientada por objetos é a utilização de *enumerated types* (*enum*). *Enum* consiste em um conjunto fixo de constantes, por exemplo, as estações do ano, os dias da semana, que traduzem grupos coesos de constantes possivelmente necessárias às aplicações.

Os valores referência propostos são capazes de indicar o que é um valor alto para NSF, o que permite identificar classes no software que estão utilizando excessivamente esse recurso ou, até mesmo, criar um mecanismo simples de detecção de *constant interface*.

5.2.10 Número de Métodos Estáticos (NSM)

O conjunto de valores para a métrica Número de Métodos Estáticos (NSM) possui melhor ajuste à distribuição *Beta*, com parâmetros $\alpha_1 = 0,017$, $\alpha_2 = 27,961$, $a = 0,000$ e $b = 1646,287$. Foram identificados os valores que representam o 90° e 96° percentil dos dados, que correspondem aos valores 1 e 3, mostrados na Tabela 5.10.

Tabela 5.10: Valores referência para NSM

Bom/Frequente	**Regular/Ocasional**	**Ruim/Raro**
$NSM \leq 1$	$1 < NSM \leq 3$	$NSM > 3$

Um método estático manipula somente atributos estáticos. Ele deve ser invocado com o nome da classe, sem a necessidade de instanciar um objeto.

Esse tipo de mecanismo quebra o conceito da programação orientada por objetos, tornado-a, de certo modo, equivalente à a programação procedural. No entanto, possui utilidade prática no contexto de desenvolvimento de software orientado por objetos na plataforma Java [Bloch, 2008]. Um dos cenários em que esse tipo de mecanismo é útil são métodos estáticos simples que retornam uma instância da classe, em vez de utilizar o construtor. Isso permite à classe ser *instance-controlled*, o que faz com que, por exemplo, uma classe garanta o comportamento *Singleton*, colocando o construtor privado. Outra utilização possível desse recurso são classes utilitárias, que são agrupamentos de métodos estáticos cuja instanciação não seria justificável. Porém, a sua utilização também possui desvantagens. A quebra do paradigma orientado por objetos representada por esse tipo de mecanismo tem consequências negativas em diversos aspectos de qualidade de software relacionados a programação orientada por objetos. Segundo Hevery [2008], métodos estáticos pioram a testabilidade dos softwares, pois testes unitários baseiam-se no conceito que um método pode ser executado e avaliado de forma independente. As dependências do método testado são simuladas com os possíveis resultados e a forma como o código avaliado responderá, independente do bom ou mau funcionamento do código dependente. Um método qualquer que depende de algum método estático tem sua testabilidade prejudicada, pois o método estático será sempre executado, violando-se a idéia de testes unitários. Quando esse método é muito complexo, o problema se maximiza. Outro ponto negativo dos métodos estáticos é que eles tornam o software menos flexível, pois eles não podem ser sobrescritos.

Os valores referência sugeridos são capazes de discriminar, dado a forma como os softwares são desenvolvidos, o que é um valor *Bom/Frequente*, *Regular/Ocasional* e *Ruim/Raro* para a métrica NSM, provendo um meio quantitativo de distinguir o que é um valor alto ou baixo para avaliar a utilização de métodos estáticos.

5.2.11 Número de Parâmetros (PAR)

O conjunto de valores para a métrica Número de Parâmetros (PAR) possui melhor ajuste à distribuição *Gumbel Max* com parâmetros $\sigma = 0,973$ e $\mu = 0,399$. Foram identificados os valores que representam o 80° e 98° percentil dos dados, que correspondem aos valores 2 e 4, sumarizados na Tabela 5.11.

Tabela 5.11: Valores referência para PAR

Bom/Frequente	**Regular/Ocasional**	**Ruim/Raro**
$PAR \leq 2$	$2 < PAR \leq 4$	$PAR > 4$

Segundo Fowler [1999], é muito difícil entender uma longa lista de parâmetros, pois elas acabam se tornando inconsistentes e difíceis de utilizar. A programação orientada por objetos permitiu que, em vez de passar por parâmetro tudo que um método precisa, seja passado o suficiente para que o método possa acessar tudo o que for necessário. Ou seja, passando um objeto em vez de tipos primitivos, a lista de parâmetros não precisa ser constantemente alterada toda vez que mais dados são ou deixam de ser necessários. Isso reduz a quantidade de parâmetros da programação orientada por objetos em relação a paradigmas tradicionais. O valor referência derivado traduz em termos estatísticos essa observação. Métodos com até dois parâmetros são os mais frequentes e representam o nível de qualidade com o qual desenvolvedores de software usualmente trabalham, provendo uma referência para a avaliação dos métodos por meio dessas métricas. O valor baixo indica que a programação orientada por objetos tende a ter uma lista de parâmetros pequena, sendo que, a faixa *Regular/Ocasional*, com valor 4, também é baixo.

5.2.12 Índice de Especialização (SIX)

O conjunto de valores para a métrica Índice de Especialização (SIX) possui melhor ajuste à distribuição *Power Function*, com parâmetros $\alpha = 0,031$, $a = 0,000$ e $b = 24,578$. Foram identificados os valores que representam o 70° e 90° percentil dos dados, que correspondem aos valores 0,019 e 1,333, sumarizados na Tabela 5.12.

Tabela 5.12: Valores referência para SIX

Bom/Frequente	**Regular/Ocasional**	**Ruim/Raro**
$SIX \leq 0,019$	$0,019 < SIX \leq 1,333$	$SIX > 1,333$

Conforme descrito na Seção 2.4, SIX visa avaliar o quanto determinada classe sobrescreve o comportamento de suas superclasses. Lorenz & Kidd [1994] sugere que

quanto maior o valor do Índice de Especialização, maior é a execução de comportamentos especializados em tempo de execução, o que torna a classe mais complexa, sugerindo que ela receba maior esforço de testes. Além disso, quando uma classe possui um alto grau de Índice de Especialização, possivelmente suas classes filhas não estarão em conformidade com a abstração da classe mãe.

Nesse sentido, identificar o que é um valor alto para SIX é de suma importância, pois é possível identificar classes com alta especialização, o que contribui com grande importância para a alteração do comportamento do software em tempo de execução. Isso torna o software mais sujeito a erros, complexo e difícil de testar. Além disso, os valores referência aqui identificados podem auxiliar a identificar classes que estejam fora de conformidade com a hierarquia de herança projetada, permitindo uma reavaliação em busca de maior qualidade de software.

5.2.13 Complexidade de *McCabe* (VG)

O conjunto de valores para a métrica Complexidade de *McCabe* (VG) possui melhor ajuste à distribuição *Chi-Squared*, com parâmetros $\nu = 1,000$ e $\gamma = 1,000$. Foram identificados os valores que representam o 70° e 90° percentil dos dados, que correspondem aos valores 2 e 4, apresentados na Tabela 5.13.

Tabela 5.13: Valores referência para VG

Bom/Frequente	**Regular/Ocasional**	**Ruim/Raro**
$VG \leq 2$	$2 < VG \leq 4$	$VG > 4$

.

De acordo com Sommerville [2010], essa métrica pode estar relacionada à facilidade de compreensão do software, no caso, no nível dos métodos. Mantendo os métodos com uma complexidade ciclomática razoável, conforme sugerido pelos valores referência derivados, espera-se que o código seja mais fácil de entender, menos susceptível a erros, mais fácil de mudar e de reutilizar.

Lopez & Habra [2005] avaliaram o limite estabelecido como aceitável para a métrica VG, de 10, sugerido pelo próprio autor da métrica, no contexto da programação orientada por objetos. Como conclusão, os autores identificaram que esse valor não era capaz de discriminar suficientemente os métodos complexos nos projetos de software orientado por objetos. Isso ocorre porque mais de 90% dos métodos avaliados possuíam uma complexidade menor que 5, e apenas 2% dos métodos possuíam uma complexidade maior que 10. Isso tornou o valor 10, sugerido pelo autor da métrica para o contexto da programação funcional, ineficiente para o contexto da programação orientada por obje-

tos. Esses resultados são concordantes com os valores referência sugeridos no presente trabalho, 2 e 4 para separar as faixas *Bom/Frequente*, *Regular/Ocasional* e *Ruim/Raro*.

5.2.14 Métodos Ponderados por Classe (WMC)

O conjunto de valores para a métrica Métodos Ponderados por Classe (WMC) possui melhor ajuste à distribuição *Log-Logistic*, com parâmetros $\alpha = 1,142$, $\beta = 4,687$ $\gamma = 0,000$. Foram identificados os valores que representam o 70° e 90° percentil dos dados, que correspondem aos valores 11 e 34, mostrados na Tabela 5.14.

Tabela 5.14: Valores referência para WMC

Bom/Frequente	**Regular/Ocasional**	**Ruim/Raro**
$WMC \leq 11$	$11 < WMC \leq 34$	$WMC > 34$

.

De acordo com Sommerville [2010], quanto maior o valor dessa métrica, mais complexa será a classe de objeto. Os objetos complexos são, provavelmente, mais difíceis de compreender e, consequentemente, de difícil manutenção. Esses objetos são, também, mais difíceis de reutilizar, pois o alto grau de complexidade prejudica a coesão da classe. Logo, ao manter os valores de WMC dentro dos padrões identificados neste trabalho, espera-se que atributos de qualidade de software, tais como facilidade de manutenção, facilidade de compreensão e reusabilidade, sejam melhorados e haja um acréscimo de qualidade interna no software. Além disso, o valor 11, sugerido no presente trabalho para classificar as classes na faixa *Bom/Frequente*, é próximo ou está dentro da faixa de aceitação estabelecida em outros trabalhos, conforme observado na Seção 3.3.

5.2.15 Profundidade de Árvore de Herança (DIT)

O Gráfico de Dispersão dos valores coletados para a métrica Profundidade de Árvore de Herança (DIT), ilustrado na Figura 5.11a, bem como o Gráfico de Frequência Relativa acumulada, Figura 5.11b, mostram uma distribuição mais uniforme dos valores coletados para esta métrica em relação ao que foi observado nas métricas CA e CE. No entanto, percebe-se que há a formação de uma cauda, mesmo que mais "achatada", denominada *fat tail*.

Como mostrado nas Figuras 5.11c e 5.11d, o conjunto de valores dessa métrica possui melhor ajuste à distribuição *Log-Logistic*, com parâmetros $\alpha = 2.170$ e $\beta = 1,469$. Essa distribuição apresenta uma curva assimétrica à direita, o que está de acordo com as observações dos Gráficos de Dispersão e Frequência Relativa Acumulada.

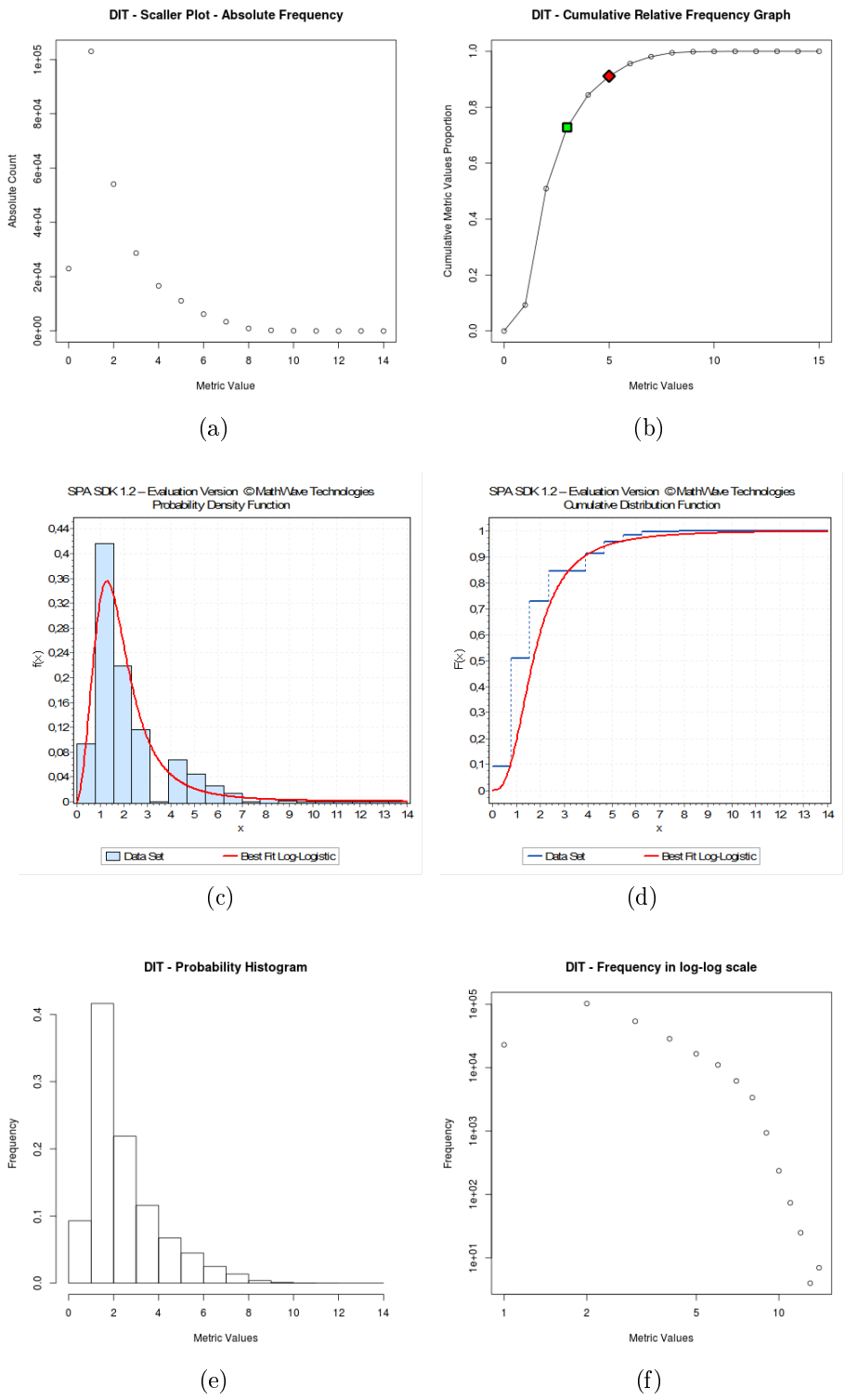


Figura 5.11: Profundidade de Árvore de Herança: (a) Gráfico de Dispersão (b) Gráfico de Frequência Relativa Acumulada (c) Função de Densidade de Probabilidade ajustada à Distribuição Log-Logistic (d) Gráfico de Frequência Relativa Acumulada ajustado à Distribuição Log-Logistic (e) Histograma de Probabilidade (f) Histograma em escala *log-log*.

A Figura 5.12 mostra a Função Densidade de Probabilidade com os parâmetros retornados, onde é possível visualizar a característica de assimetria à direita da distribuição com esses parâmetros.

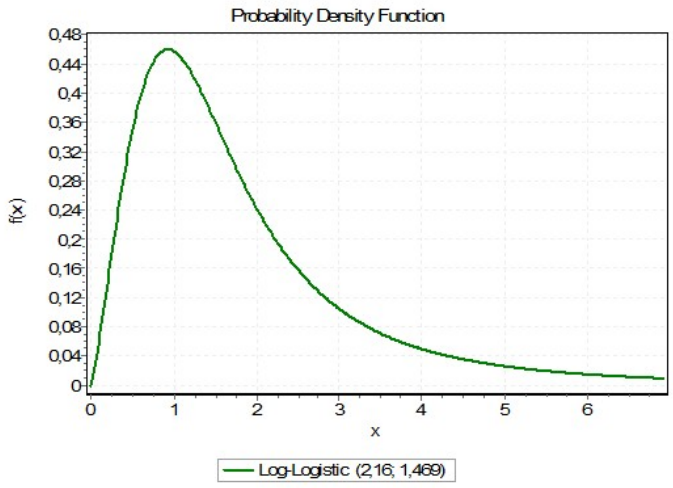


Figura 5.12: DIT - Função de Densidade de Probabilidade com os parâmetros retornados pela ferramenta *EasyFit*

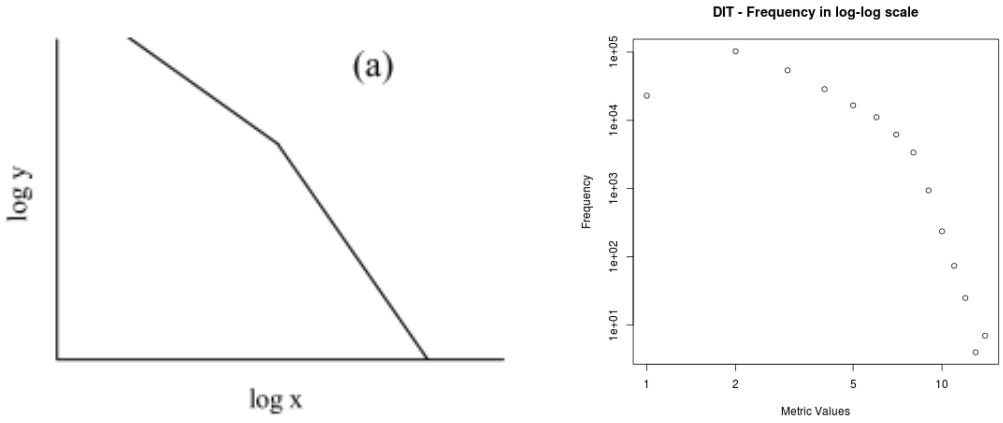
Pelo Histograma de Probilidade ilustrado na Figura 5.11e, observa-se também a formação da cauda, mesmo que mais achatada do que nas métricas CA e CE. Os dados em escala logarítmica, mostrados na Figura 5.11f, não seguem estritamente uma lei de potência. No entanto, apresentam duas retas inclinadas à esquerda com um ponto de junção, característica de uma dupla lei de potência. Em uma dupla lei de potência há duas faixas de valores com diferentes expoentes, conforme apontado no trabalho de Milojević [2010]. Essa situação é considerada como um "desvio" de uma lei de potência pura, podendo ser ocasionado, inclusive, por "ruídos" nos dados. Na Figura 5.13, é possível observar a semelhança entre o desvio relatado no trabalho de Milojević [2010] e os dados em escala logarítmica dessa métrica.

Foram identificados os valores que representam o 70° e 90° percentil dos dados, que correspondem aos valores 2 e 4. Isso significa que 70% dos pacotes possuem $DIT \leq 2$ e 90% dos pacotes possuem $DIT \leq 4$. Os valores 70% e 90% foram identificados por meio de uma análise visual no Gráfico de Frequência Relativa Acumulada da Figura 5.11b. Os pontos marcados com a cor verde / quadrado e vermelha / losango representam as regiões identificadas nessa análise. A Tabela 5.15 sumariza os valores referência derivados para a métrica Profundidade de Árvore de Herança (DIT).

Esse resultado está de acordo com aquele obtido no trabalho de Ferreira et al. [2012]. Os autores sugerem para a métrica DIT um valor médio representativo, considerando a maior distância de uma classe à raiz de sua árvore de herança como 2, em

Tabela 5.15: Valores referência para DIT

Bom/Frequente	**Regular/Ocasional**	**Ruim/Raro**
$DIT \leq 2$	$2 < DIT \leq 4$	$DIT > 4$



(a) Lei de Potência Dupla. Fonte: Milojević [2010] (b) Histograma em escala *log-log* DIT

Figura 5.13: Comparação Lei de Potência Dupla e dados de DIT em escala logarítmica

média, mesmo valor derivado nesse trabalho para a faixa *Bom/Frequente*. No entanto, nesta pesquisa, foram identificadas também as faixas *Regular/Ocasional* e *Ruim/Raro*. Isso pode ser explicado pelo fato de que a amostra que está sendo utilizada nesse trabalho é significativamente maior e possui valores maiores de medição da métrica, mesmo que pouco frequentes, mas significantes na formação da cauda. Além disso, utilizou-se outra distribuição como melhor encaixe para os valores da métrica, que possui assimetria à direita e, consequentemente, o valor típico não é a média. Isso permitiu identificar faixas de valores *Bom/Frequente*, *Regular/Ocasional* e *Ruim/Raro*.

De acordo com Sommerville [2010], quanto maior for a profundidade da árvore de herança, mais complexo será o projeto. O quanto mais profunda uma classe estiver em uma hierarquia de classes, mais métodos serão herdados e, consequentemente, a classe se tornará mais complexa. O valor referência derivado para a faixa *Bom/Frequente* traduz uma árvore de herança rasa, com no máximo dois níveis, o que mantém as classes filhas próximas da classes raízes na hierarquia de herança.

5.2.16 Ausência de Coesão em Métodos (LCOM)

O Gráfico de Dispersão dos valores coletados para a métrica Ausência de Coesão em Métodos (LCOM), na Figura 5.14a, apresenta uma característica interessante, pois há uma queda brusca de contagem de valores entre 0 e 0,1. Há 186.162 classes com

$LCOM = 0$, representando 75% dos dados, e, para $LCOM = 0,1$, há 1075 classes, representando 0,004% dos dados. Há uma queda abrupta na quantidade, e o gráfico fica sem um aspecto visual que traduza uma cauda-pesada.

No Gráfico de Frequência Relativa Acumulada ilustrado na Figura 5.14b, percebemos que a distribuição sugere um formato de cauda, no entanto, em sua curva de aproximação dos 100%, há um comportamento distinto de amostras de dados mais aderentes a característica de cauda-pesada. A aproximação dos 100% torna-se lenta a partir dos 75%, retomando o crescimento a partir dos 80% de frequência relativa acumulada.

Como mostrado nas Figuras 5.14c e 5.14d, o conjunto de valores dessa métrica possui melhor ajuste à distribuição *Beta*, com parâmetros $\alpha_1 = 0,043$, $\alpha_2 = 6,777$, $a = 0,000$ e $b = 8,290$. Pela linha do ajuste dos dados à distribuição, é possível observar a característica de cauda-pesada, de forma ressaltada. A Figura 5.15 mostra a Função Densidade de Probabilidade com os parâmetros retornados, onde é possível visualizar a característica de cauda-pesada da distribuição com esses parâmetros.

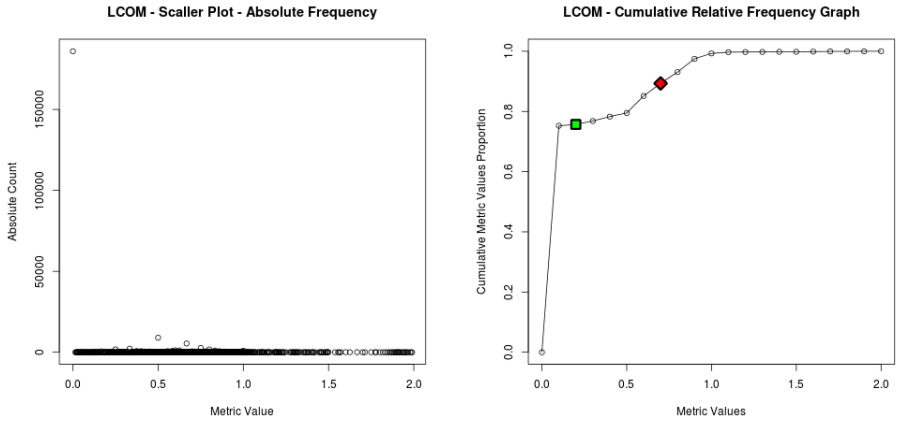
Percebe-se pela Figura 5.14e o mesmo que foi visualizado no Gráfico de Dispersão, uma queda abrupta na frequência que descaracteriza a cauda. O Histograma em escala *log-log* mostrado na Figura 5.14f apresenta pontos dispersos não característicos de uma lei de potência. Porém, nem toda distribuição de cauda-pesada é uma lei de potência, como observado no trabalho de Baxter et al. [2006].

Foram identificados os valores que representam o 75.5° e 90° percentil dos dados, que correspondem aos valores 0,167 e 0,725. Isso significa que 75.5% dos pacotes possuem $LCOM \leq 0,167$ e 90% dos pacotes possuem $LCOM \leq 0,725$. Os valores 75.5% e 90% foram identificados por meio de uma análise visual no Gráfico de Frequência Relativa Acumulada da Figura 5.14b. Os pontos marcados com a cor verde / quadrado e vermelha / losango representam as regiões identificadas nessa análise. A Tabela 5.16 sumariza os valores referência derivados para a métrica Ausência de Coesão em Métodos (LCOM).

Tabela 5.16: Valores referência para LCOM

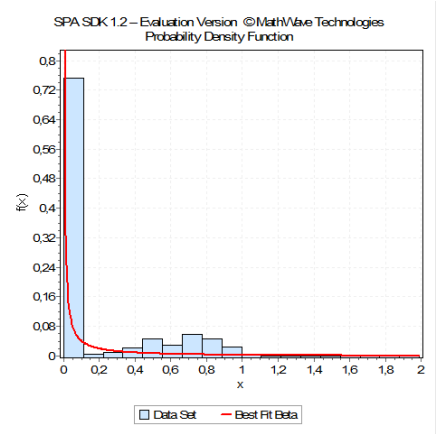
Bom/Frequente	**Regular/Ocasional**	**Ruim/Raro**
$LCOM \leq 0,167$	$0,162 < LCOM \leq 0,725$	$LCOM > 0,725$

A coesão facilita a compreensão, reúso e manutenção do código [Chidamber & Kemerer, 1994; Fowler, 1999; Sommerville, 2010]. LCOM mede a ausência de coesão, então, quanto maior for o valor da métrica, a classe apresenta menos coesão. Um alto valor de LCOM indica que a classe pode ter muitas tarefas diferentes para realizar,

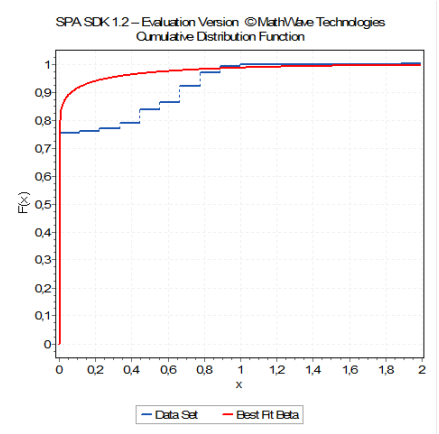


(a)

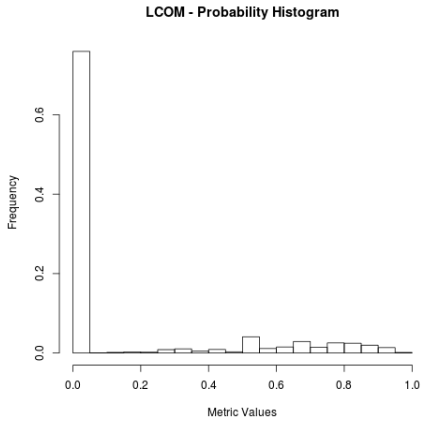
(b)



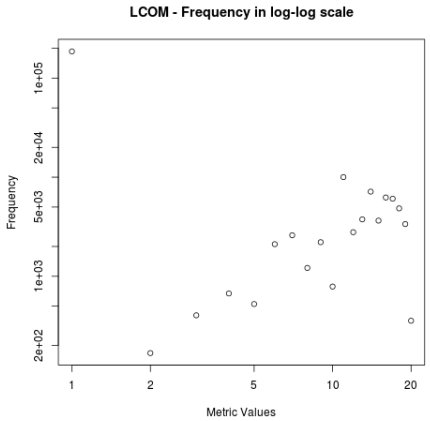
(c)



(d)



(e)



(f)

Figura 5.14: Ausência de Coesão em Métodos: (a) Gráfico de Dispersão (b) Gráfico de Frequência Relativa Acumulada (c) Função de Densidade de Probabilidade ajustada à Distribuição Beta (d) Gráfico de Frequência Relativa Acumulada ajustado à Distribuição Beta (e) Histograma de Probabilidade (f) Histograma em escala *log-log*.

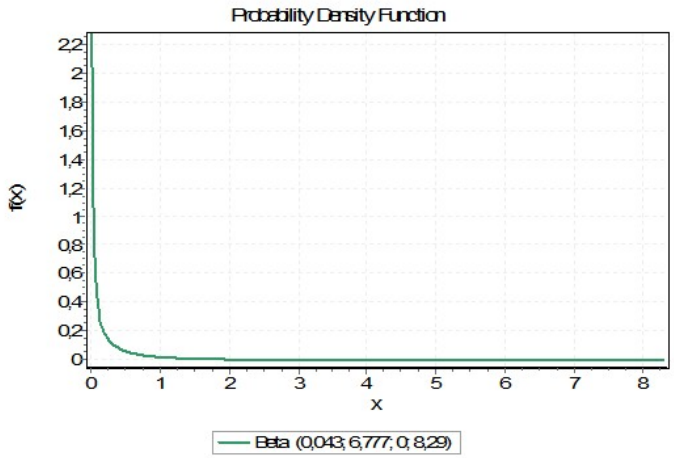


Figura 5.15: LCOM - Função de Densidade de Probabilidade com os parâmetros retornados pela ferramenta *EasyFit*

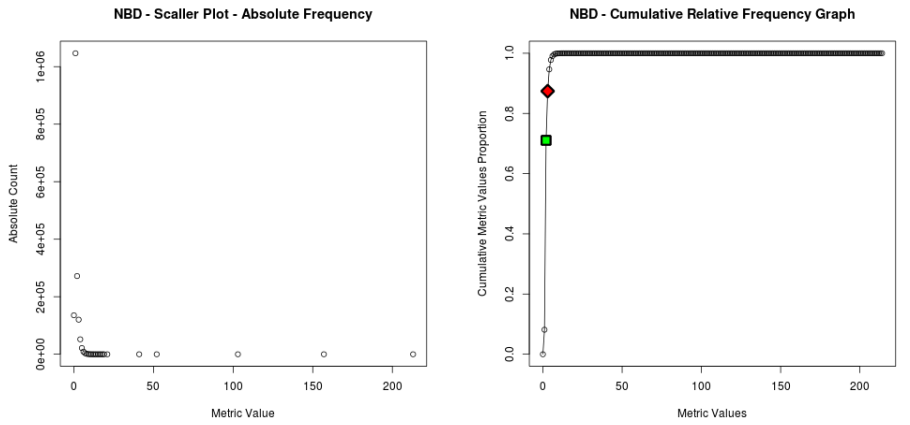
sendo melhor utilizar objetos mais específicos, dividindo-a em classes menores e que apresentem maior coesão. Alshayeb [2009] avaliou o impacto de refatoração em métricas de coesão, dentre as quais, LCOM. Os resultados mostram que houve aumento da coesão com a refatoração do código. Isso sugere a validade dos valores referência aqui identificados para avaliar classes potencialmente problemáticas dentro dos softwares orientados por objetos.

Os valores referência sugeridos, baseado nos padrões de qualidade com que os softwares orientados por objetos tem sido desenvolvidos, indicam aos desenvolvedores o que é um valor alto de LCOM, permitindo identificar classes que possam ser refatoradas de forma a melhorar a qualidade dos softwares.

5.2.17 Profundidade de Blocos Aninhados (NBD)

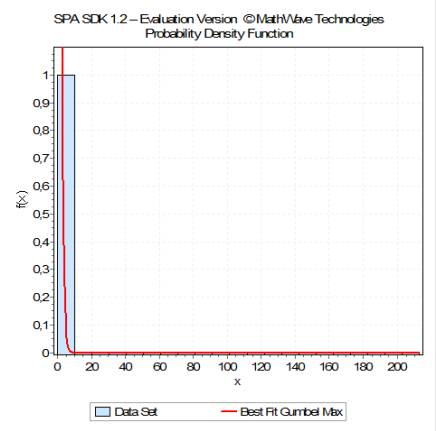
O Gráfico de Dispersão dos valores coletados para a métrica Profundidade de Blocos Aninhados (NBD), na Figura 5.16a, sugere uma distribuição de cauda-pesada. A Figura 5.16b ressalta ainda mais essa característica, pois permite observar que a aproximação dos 100% de frequência acumulada ao longo do eixo x (valores das métricas) ocorre rapidamente. Isso significa que há um grande número de métodos com uma quantidade pequena de blocos aninhados, e um pequeno número de métodos com uma quantidade grande de blocos aninhados.

Como mostrado nas Figuras 5.16c e 5.16d, o conjunto de valores dessa métrica possui melhor ajuste à distribuição *Gumbel Max*, com parâmetros $\sigma = 0,858$ e $\mu = 0,931$. Essa distribuição apresenta uma curva assimétrica à direita, o que está de acordo com as observações dos Gráficos de Dispersão e Frequência Relativa Acumu-

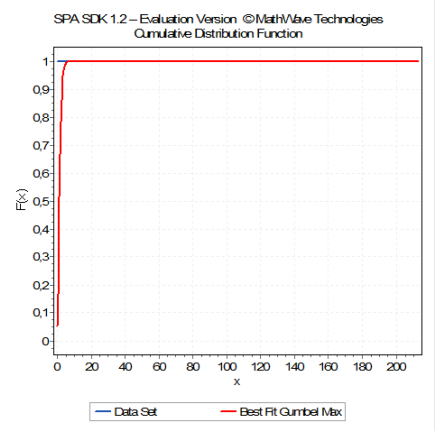


(a)

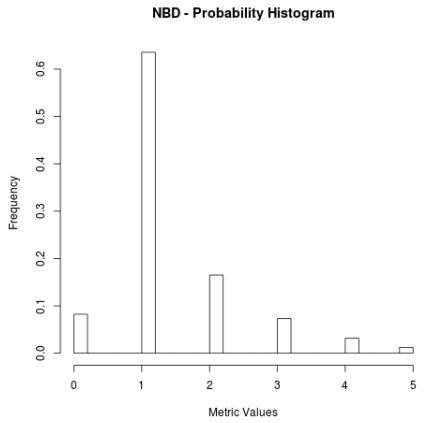
(b)



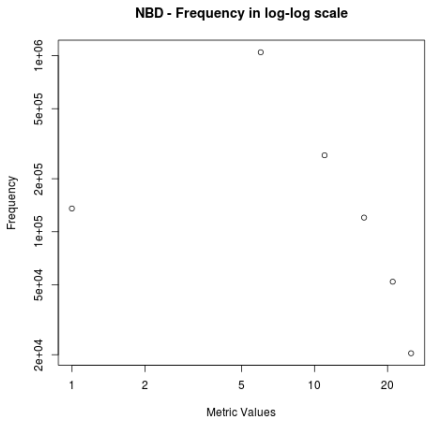
(c)



(d)



(e)



(f)

Figura 5.16: Profundidade de Blocos Aninhados: (a) Gráfico de Dispersão (b) Gráfico de Frequência Relativa Acumulada (c) Função de Densidade de Probabilidade ajustada à Distribuição Gumbel Max (d) Gráfico de Frequência Relativa Acumulada ajustado à Distribuição Gumbel Max (e) Histograma de Probabilidade (f) Histograma em escala *log-log*.

lada. A Figura 5.17 mostra a Função Densidade de Probabilidade com os parâmetros retornados, onde é possível visualizar a característica de assimetria à direita da distribuição com esses parâmetros.

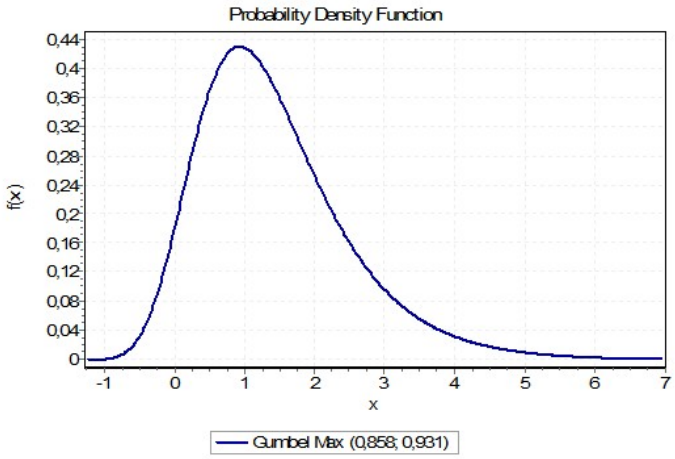
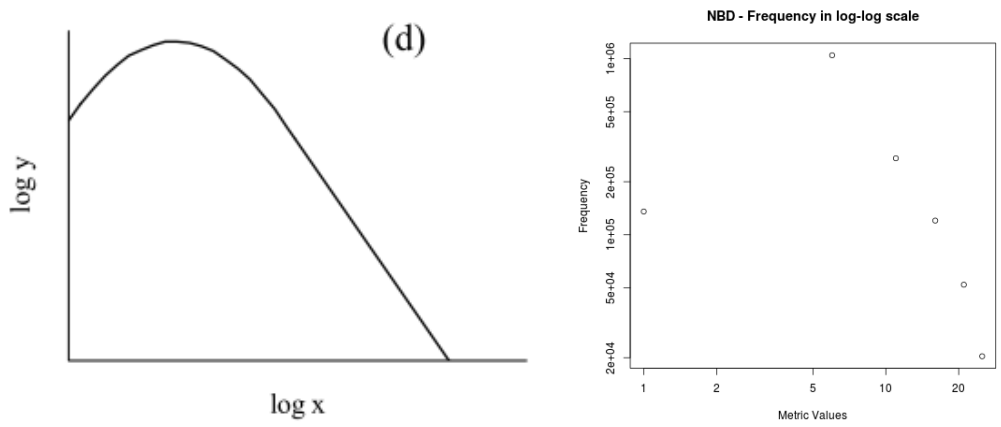


Figura 5.17: NBD - Função de Densidade de Probabilidade com os parâmetros retornados pela ferramenta *EasyFit*

A Figura 5.16e mostra o histograma de probabilidade dos dados. Percebe-se que a terceira barra, que representa o valor de 2 de NBD, constitui menos de 20% dos dados, enquanto a quarta barra, que representa o valor 3, constitui menos de 10% dos dados. A Figura 5.16f mostra esses dados em escala logarítmica. Percebe-se, excluindo o primeiro ponto, pois há um crescimento discordante do restante dos dados entre 0 e 1, algo semelhante a uma reta inclinada à esquerda, característica de uma lei de potência. Se for analisado outro "desvio" da lei de potência relatado no trabalho de Milojević [2010], é possível identificar uma semelhança considerável. Nesse "desvio", a lei de potência ocorre para valores maiores de x, enquanto que uma distribuição log-normal (ou semelhante) se encaixa melhor para valores menores de x. Observa-se na Figura 5.18 a semelhança entre o desvio relatado no trabalho de Milojević [2010] e os dados em escala logarítmica dessa métrica.

Foram identificados os valores que representam o 70° e 90° percentil dos dados, que correspondem aos valores 1 e 3. Isso significa que 70% dos métodos possuem $NBD \leq 1$ e 90% dos métodos possuem $NBD \leq 3$. Os valores 70% e 90% foram identificados por meio de uma análise visual no Gráfico de Frequência Relativa Acumulada da Figura 5.16b. Os pontos marcados com a cor verde / quadrado e vermelha / losango representam as regiões identificadas nessa análise. A Tabela 5.17 sumariza os valores referência derivados para a métrica Profundidade de Blocos Aninhados (NBD).

De acordo com Sommerville [2010], declarações profundamente aninhadas são



(a) Distribuição Log-Normal/Lei de Potência. (b) Histograma em escala *log-log* NBD
Fonte: Milojević [2010]

Figura 5.18: Comparação Lei de Potência Dupla e dados de NBD em escala logarítmica

Tabela 5.17: Valores referência para NBD

Bom/Frequente	**Regular/Ocasional**	**Ruim/Raro**
$NBD \leq 1$	$1 < NBD \leq 3$	$NBD > 3$

difíceis de compreender e são potencialmente propensas a erros. Logo, ao manter os métodos com um valor baixo de NBD, tal como o valor sugerido ≤ 1 (para a faixa *Bom/Frequente*), espera-se que os métodos tornem-se mais legíveis e menos propensos a erros. Nesse trabaho sugere-se que, acima de 3, o método seja classificado como *Ruim/Raro*, sugerindo que ele seja refatorado, dividindo-o em dois ou mais métodos.

5.2.18 Distância Normalizada (RMD)

O Gráfico de Dispersão dos valores coletados para a métrica Distância Normalizada (RMD), ilustrado Figura 5.19a, apresenta uma característica interessante, pois há uma quedra brusca de contagem de valores entre 0 e os valores menores que 0,2, e, a posterior, observa-se algumas ocorrências de valores maiores que 0,2 em uma quantidade um pouco maior do que os valores anteriores. No entanto, é pouco em relação ao valor inicial 0, que caracteriza a maioria das ocorrências dessa métrica nos pacotes de software.

No Gráfico de Frequência Relativa Acumulada exibido na Figura 5.19b, percebe-se que a distribuição sugere um formato de cauda, no entanto, em sua curva de aproximação dos 100%, há um comportamento distinto de amostras de dados mais aderentes a característica de cauda-pesada, tal como ocorreu na métrica RMA. A aproximação

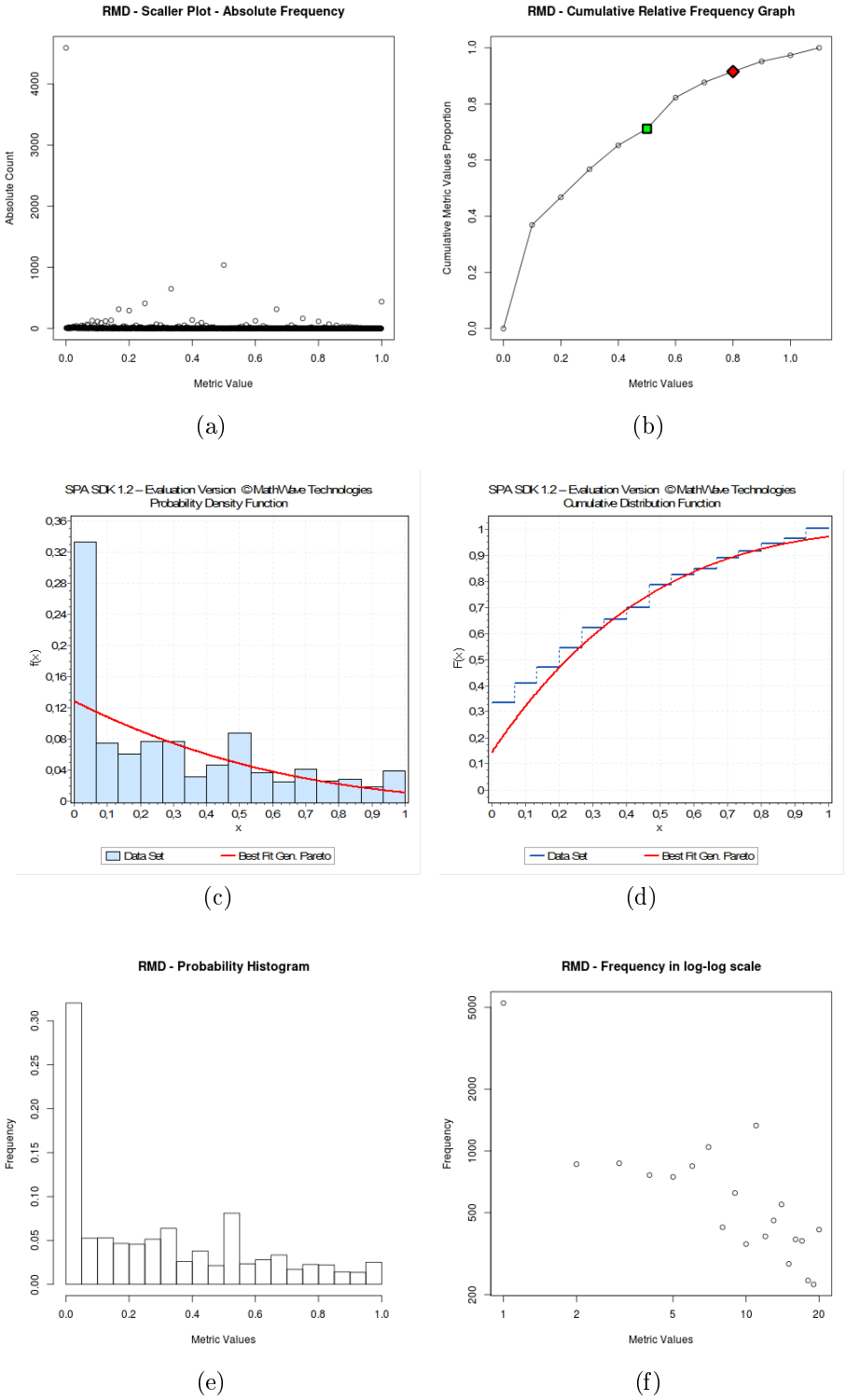


Figura 5.19: Distância Normalizada: (a) Gráfico de Dispersão (b) Gráfico de Frequência Relativa Acumulada (c) Função de Densidade de Probabilidade ajustada à Distribuição Gen. Pareto (d) Gráfico de Frequência Relativa Acumulada ajustado à Distribuição Gen. Pareto (e) Histograma de Probabilidade (f) Histograma em escala *log-log*.

dos 100% torna-se lenta a partir dos 40%, aumentado a área sobre a curva.

Como mostrado nas Figuras 5.19c e 5.19d, o conjunto de valores dessa métrica possui melhor ajuste à distribuição *Generalized Pareto*, com parâmetros $k = -0,256$, $\sigma = 0,464$ e $\mu = -0,071$. Essa distribuição tem característica de cauda-pesada. A Figura 5.20 mostra a Função Densidade de Probabilidade com os parâmetros retornados, onde é possível visualizar a característica de cauda-pesada da distribuição com esses parâmetros.

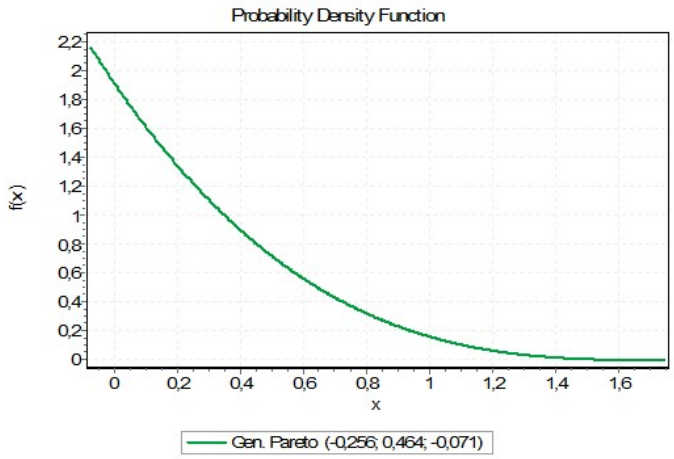


Figura 5.20: RMD - Função de Densidade de Probabilidade com os parâmetros retornados pela ferramenta *EasyFit*

A Figura 5.19e mostra o histograma de probabilidade dos dados. Percebe-se que a segunda barra, que representa os valores de RMD entre 0,05 e 0,1, constitui 0,05% dos dados. O restante das barras varia positivamente ou negativamente em torno desses valores. A Figura 5.19f mostra esses dados em escala logarítmica. Percebe-se uma linearidade dos pontos em relação a uma reta inclinda à esquerda, mas os pontos estão um pouco dispersos ao redor dessa reta. No entanto, nem toda distribuição de cauda-pesada é uma lei de potência, como observado no trabalho de Baxter et al. [2006].

Foram identificados os valores que representam o 70° e 90° percentil dos dados, que correspondem aos valores 0,467 e 0,750. Isso significa que 70% dos pacotes possuem $RMD \leq 0,467$ e 90% dos pacotes possuem $RMD \leq 0,750$. Os valores 70% e 90% foram identificados por meio de uma análise visual no Gráfico de Frequência Relativa Acumulada da Figura 5.19b. Os pontos marcados com a cor verde / quadrado e vermelha / losango representam as regiões identificadas nessa análise. A Tabela 5.18 sumariza os valores referência derivados para a métrica Distância Normalizada (RMD).

Conforme explicitado na Seção 2.3, essa métrica propõe uma noção de balanceamento entre Instabilidade e Abstração [Martin, 1994]. Um pacote pode ser parcial-

Tabela 5.18: Valores referência para RMD

Bom/Frequente	**Regular/Ocasional**	**Ruim/Raro**
$RMD \leq 0,467$	$0,467 < RMD \leq 0,750$	$RMD > 0,750$

.

mente extensível se ele for parcialmente abstrato. Quaisquer valores que estejam na *Main Sequence* representam um balanceamento aceito por Martin como equilibrado. Quanto menor for o valor, menos distante o pacote está da *Main Sequence*, e, dessa forma, espera-se uma relação mais equilibrada entre Instabilidade e Abstração.

De acordo com a definição da métrica, uma relação mais equilibrada sugere um grau de balanceamento aceitável de classes concretas e abstratas em relação a seus acoplamentos aferentes e eferentes, de forma a se obter um projeto mais estável, que seja menos sensível a mudanças evolutivas no software e represente um avanço em termos de qualidade de software. Nesse sentido, os valores referência aqui sugeridos são capazes de discriminar um valor alto, que representa a faixa *Ruim/Raro,* de um valor baixo, que representa a faixa *Bom/Frequente* da métrica RMD. Acredita-se que esses valores possam ser úteis para avaliar o equilíbrio do pacote e que identificar pacotes com valores altos dessa métrica pode ter importância na avaliação arquitetural do projeto, buscando evoluir e melhor equilibrar a divisão de responsabilidades entre os pacotes do software.

5.3 Catálogo de Valores Referência

Nesta seção são sumarizados os valores referência derivados para cada uma das métricas, Tabela 5.19, que constitui um catálogo de valores referência para métricas de softwares orientados por objetos. Cada linha da tabela contém a identificação da métrica, bem como o valor estabelecido para as Faixas *Bom/Frequente*, *Regular/Ocasional* e *Ruim/Raro*. A Tabela 5.20 cataloga as distribuições estatísticas que possuem melhor encaixe com os valores coletados das métricas.

Tabela 5.19: Catálogo de valores referência para métricas de softwares orientados por objetos

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	$RMD \leq 0,467$	$0,467 < RMD \leq 0,750$	$RMD > 0,750$

5.4 Discussão

Esta seção apresenta uma discussão acerca de aspectos relacionados ao método utilizado e sua aplicação para a derivação dos valores referência. O objetivo é avançar na discussão acerca de suas limitações e benefícios dentro do panorama atual da área.

Tabela 5.20: Catálogo de distribuições estatísticas que melhor descrevem as métricas estudadas.

Métrica	Distribuição	Parâmetros
CA	*Gen. Extreme Value*	$k = 0.797$, $\sigma = 4.303$, $\mu = 1.775$
CE	*Gen. Extreme Value*	$k = 0.527$, $\sigma = 2.834$, $\mu = 2.397$
MLOC	*Pareto 2*	$\alpha = 1.226$, $\beta = 3.051$
NOC	*Log-Logistic*	$\alpha = 1.452$, $\beta = 5.520$
NOF	*Beta*	$\alpha_1 = 0.059$, $\alpha_2 = 64.580$, $b = 2026.446$
NOM	*Weibull*	$\alpha = 0.852$, $\beta = 5.879$
NORM	*Power Function*	$\alpha = 0.006$, $a = 0$, $b = 235.200$
NSC	*Beta*	$\alpha_1 = 0.001$, $\alpha_2 = 9.467$, $b = 25465.529$
NSF	*Beta*	$\alpha_1 = 0.018$, $\alpha_2 = 18.284$, $b = 4130.615$
NSM	*Beta*	$\alpha_1 = 0.017$, $\alpha_2 = 27.961$, $b = 1646.287$
PAR	*Gumbel Max*	$\sigma = 0.973$, $\mu = 0.399$
SIX	*Power Function*	$\alpha = 0.031$, $a = 0$, $b = 24.578$
VG	*Chi-Squared*	$\nu = 1.000$, $\gamma = 1.000$
WMC	*Log-Logistic*	$\alpha = 1.142$, $\beta = 4.687$, $\gamma = 0.000$
DIT	*Log-Logistic*	$\alpha = 2.170$, $\beta = 1,469$
LCOM	*Beta*	$\alpha_1 = 0.043$, $\alpha_2 = 6.777$, $b = 8.290$
NBD	*Gumbel Max*	$\sigma = 0.858$, $\mu = 0.931$
RMD	*Generalized Pareto*	$k = -0.256$, $\sigma = 0.464$, $\mu = -0.071$

5.4.1 Interpretação do Valor da Métrica

Conforme relatado por Bouwers et al. [2012], conduzir alterações nos softwares com o objetivo de melhorar o valor da métrica é uma atividade puramente "cosmética". Dessa forma, o tratamento do valor leva a refatorações que simplesmente "agradam às métricas", o que é um desperdício sério de recursos.

Bouwers et al. utilizam um exemplo interessante para ilustrar essa situação. Considere um projeto no qual um método qualquer é identificado como detentor de uma quantidade excessiva de parâmetros. Isso pode indicar que o método está implementando diferentes funcionalidades e dividi-lo em métodos menores, que implementariam uma única funcionalidade, tornaria mais fácil o entendimento.

Um segundo problema que pode estar ocorrendo com esse método é a falta de um objeto necessário ao projeto que agrupe parâmetros que são comumente utilizados em conjunto. Por exemplo, considere um método que recebe como parâmetro dois objetos do tipo *Date* chamados *startDate* e *finalDate*. Esses nomes sugerem que esses dois parâmetros poderiam formar um objeto do tipo *DatePeriod*, no qual a *startDate* deve ser anterior a *endDate*. Quando vários métodos recebem esses dois parâmetros como entrada, a introdução do objeto do tipo *DatePeriod* no domínio pode ser de grande

valor à qualidade do projeto, tornando-o mais legível e diminuindo possíveis esforços de manutenção.

Essas duas situações relatadas, nas quais a indicação do aspecto quantitativo implicou em melhorias reais no projeto, tornando-o mais legível e, consequemente, de melhor manutenabilidade, atingiram o objetivo da organização, que é ter um software de maior qualidade.

No entanto, o que um programador pode fazer é tratar diretamente o valor da métrica. Por exemplo, ele pode mover os parâmetros do método para atributos da classe ou, até mesmo, substituí-los por um objeto do tipo *Map*, que é uma lista de elementos do tipo *key-value*, na qual mapeia-se uma *string* (*key*, que representa o parâmetro) com o seu valor (*value*, que representa o valor do parâmetro).

Todas as estratégias citadas são capazes de reduzir o número de parâmetros dos métodos. Mas, obviamente, se o objetivo final é melhorar a legilibilidade do código e reduzir futuros esforços de manutenção, as soluções que visam ao tratamento dos sintomas em vez do tratamento das causas do problema não terão real efetividade na melhoria da qualidade do software. Esse tipo de situação pode acontecer pelo simples fato de que os desenvolvedores não estão cientes de que há objetivos maiores ao aplicar as métricas no gerenciamento da qualidade interna dos softwares.

Quando se fala em manipulação de valores, é importante ressaltar que os valores referência derivados e propostos neste trabalho possuem um claro mapeamento do valor numérico para uma variável qualitativa ordinal, que é o nome da faixa. Sobre o conceito qualidade/frequência estabelecido, focou-se todo o processo, desde a sua concepção inicial, no trabalho de Ferreira et al. [2012], passando pela derivação em si de uma grande quantidade de métricas que foi exposta neste capítulo e as avaliações que serão realizadas. Tudo isso gera a possibilidade de utilização das variáveis ordinais em vez da medida em si para quando da sua utilização.

Quando se vislumbra a efetiva aplicação do catálogo de valores referência proposto neste trabalho de pesquisa, não é almejado, por exemplo, que ferramentas que o utilizem expressem para os desenvolvedores número absolutos. É importante que o programador interprete adequadamente que aquele método, classe ou pacote possui problemas estruturais de acordo com um determinado atributo de qualidade mapeado pela métrica. E mais do que isso, conforme relatado por Bouwers et al. [2012], o programador deve ter a consciência do objetivo de qualidade almejado por aquele controle. A utilização de uma variável ordinal que reflete a classificação qualitativa possibilita que o programador não se paute por um número e sim pela qualidade. O número não deve ser o objetivo final, o número deve ser um meio para se chegar ao objetivo final, que é o aumento da qualidade de software.

5.4.2 Unidirecionalidade dos Valores Referência propostos em Relação à Qualidade

Alguns podem dizer que a abordagem adotada neste trabalho é unidirecional em relação à qualidade, pois assume que existe uma clara orientação em termos da qualidade para menores valores da métrica. De fato, essa presunção existe. Porém, há de ser observado que esse problema inicia-se na própria definição das métricas. Todas as métricas estudas, mesmo aquelas para as quais não foram sugeridos valores referência, possuem uma definição muito mais clara em termos de qualidade deteriorada para altos valores do que para baixos valores.

Um exemplo clássico desse "padrão" adotado de que quanto maior o valor da métrica, pior é o aspecto avaliado, é métrica a LCOM. Uma classe deve ter alta coesão. Mas, para manter esse sentido de orientação de que valores altos são piores, Chidamber & Kemerer [1994] definiram a métrica como ausência de coesão. Perguntas do tipo: (1) Qual é o valor mínimo de acoplamento que um pacote deve possuir (não faz sentido um pacote totalmente desacoplado do projeto)? (2) Qual é o valor mínimo de atributos que uma classe deve possuir?, e, (3) Qual é o valor mínimo de métodos de uma classe?, são, de fato, muito mais problemáticas tanto em termos de definição de métricas como em definição de valores referência.

Nesse sentido, acredita-se que a faixa *Bom/Frequente* reflete, além de ser a prática comum no desenvolvimento de software, o que é consensual em termos de qualidade. Naturalmente existem problemas relacionados a esse tipo de definição, mas eles serão muito mais relacionados a forma como o catálogo de valores referência será aplicado do que aos próprios valores.

Um exemplo claro desse cenário é a métrica DIT, que possui uma orientação clara em relação a que altos valores caracterizam uma hierarquia de herança profunda, na qual mais métodos serão herdados e, consequentemente, a classe se tornará mais complexa. Além disso, grandes distâncias entre classes filhas e a classe na raíz na hierarquia de herança caracteriza um projeto mais complexo e, consequentemente, mais propenso a erros. Essa orientação é clara no sentido de que árvores de herança não devem ser profundas. Analisando o trabalho de Chidamber & Kemerer [1994] na definição da métrica, vê-se que os autores sugerem que muitas classes com DIT muito pequeno sugerem um projeto "top heavy" (muitas classes estão perto da raíz), o que pode indicar que o projeto pode não estar tirando vantagem do reúso de métodos pela herança. No entanto, os autores da métrica DIT colocam que isso pode ocorrer em função da aplicação e é uma conclusão que transcende a responsabilidade da métrica em avaliar a profundidade da árvore de herança da classe e, consequentemente, do valor

referência para ela proposto.

Trazendo essa situação relatada por Chidamber & Kemerer [1994] para um cenário em que o valor referência para DIT, proposto neste trabalho, é utilizado, se todas as classes do sistema mostrarem, por exemplo, $DIT = 0$, todas elas seriam classificadas como *Bom/Frequente*. Porém, o projeto seria "top-heavy", sugerindo uma utilização subótima da herança. Essa conclusão é, provavelmente, certa, mas contraria o que foi proposto como valor referência para DIT e os bons resultados mostrados nesse exemplo. Essa é uma conclusão que emerge de uma observação geral do projeto do software e que não tem qualquer relação com o valor referência proposto ter classificado as classes como *Bom/Frequente* de forma errada. De fato, uma classe com uma árvore de herança rasa ($DIT = 0$) não apresenta os problemas listados pelos autores de DIT quando da definição da métrica. Além disso, é o valor mais frequente da prática de desenvolvimento de software.

Então, pode-se pensar que o fato de todas as classes terem sido bem avaliadas é um problema de definição do valor referência, afinal, o projeto não utiliza a herança da forma apropriada. Porém, isso seria, na verdade, um problema de entendimento da definição da métrica e de aplicação do catálogo proposto. Nenhuma das métricas para as quais foram propostos valores referência avaliam a utilização do recurso correspondente como um todo dentro do software. Isso não significa que conclusões possam emergir da observação da classificação geral dos artefatos, mas que esse tipo de análise deve ser realizada de forma cuidadosa.

Essas considerações destacam algumas questões relacionadas à utilização do catálogo proposto e a interpretação dos resultados obtidos. É fundamental que os cenários em que os valores referência forem aplicados sejam avaliados de forma a minimizar interpretações equivocadas.

5.4.3 Análise de Sensibilidade do Método Relativa à Escolha do Corpus

A avaliação de sensibilidade do método utilizado para derivação dos valores referência em relação ao corpus [Tempero et al., 2010] foi conduzida com um alto nível de sensibilidade na própria definição do método [Ferreira et al., 2012]. Os autores aplicaram o método não somente em todo o conjunto de sistemas utilizados, mas também agrupando-o nos seguintes subconjuntos: por aplicação, por domínio e por tipo. Ou seja, foram consideradas não somente o conjunto inteiro de sistemas, como também subconjuntos dele. Como resultado, Ferreira et al. não identificaram diferenças relevantes nos valores referência entre essas abordagens. Em acordo com esses resultados

e com o objetivo principal deste trabalho de dissertação, que é propor um catálogo de valores referência para métricas de softwares orientados por objetos, aplicou-se o método no conjunto inteiro de softwares disponíveis.

5.4.4 Tamanho da Amostra

Uma preocupação importante em relação à aplicação do método de derivação de valores referência é o tamanho da amostra quando da aplicação do método, principalmente para o *data-fitting* descrito na Seção 4.1.2.2. De fato, o tamanho da amostra tem influência significativa na escolha da distribuição apropriada para os dados e pode colocar em dúvida os resultados quando aplicadas a poucas medições. Muitas distribuições se distinguem apenas quando presentes valores extremamente altos e, por consequência, infrequentes. O método utilizado nesta dissertação foi aplicado a um conjunto de softwares equivalente ou maior do que os trabalhos que tem sido realizados, totalizando mais de 16.000 pacotes, mais de 247.000 classes e mais de 1.600.000 métodos.

5.4.5 Sensibilidade ao Teste Estatístico e Escolha da Distribuição

O método proposto por Ferreira et al. [2012] utiliza do teste de *Kolmogorov-Smirnoff* para selecionar a distribuição apropriada aos dados. A importância do estabelecimento da distribuição reside no fato dela estabelecer as características da curva, possibilitando a definição acerca da representatividade da média ou se a distribuição segue uma cauda-pesada ou possui assimetria à direita. No entanto, seria possível definir essa característica sem estabelecer a distribuição apropriada aos dados, por meio de outros gráficos acessórios gerados. Fez-se isso pelo fato de que é uma informação importante e que traz mais segurança ao método. Se outro teste fosse aplicado, por exemplo, *Anderson-Starling*, ele pode retornar uma distribuição diferente, mas as características extraídas acerca da distribuição selecionada necessárias ao processo de derivação serão mantidas, pois como os dados são os mesmos, a distribuição pode variar, mas ela continuará retratando a não representividade da média, se for o caso.

5.4.6 Detalhes de Implementação das Métricas

As métricas analisadas neste trabalho foram coletadas utilizando o plugin *Metrics*. Diferenças de implementação de métricas de software que ocorrem por interpretações distintas entre as ferramentas de extração são comuns devido a falta de definições es-

pecíficas. Detalhes de implementação das métricas utilizadas estão bem documentadas neste trabalho e estão presentes no próprio sítio do *plugin Metrics*[2].

5.5 Conclusão

Neste capítulo foi apresentado o processo de derivação dos valores referência sugeridos para 18 métricas de softwares orientados por objetos, culminando com a criação de um catálogo que mostra os resultados obtidos e serve como um *benchmark* para a avaliação quantitativa da qualidade interna de softwares dessa natureza. Além disso, foi apresentado um catálogo com as distribuições estatísticas seguidas pelos conjuntos de medidas de cada uma dessas métricas, informação resultante do próprio processo de derivação e que permite identificar características acerca de como os softwares estão sendo construídos em relação aos aspectos avaliados pela métrica. O capítulo também trouxe uma discussão acerca dos benefícios e limitações do método utilizado no processo de derivação, explorando aspectos relacionados à validade dos resultados obtidos e suas utilizações.

No capítulo seguinte será apresentada a ferramenta *RAFTool* (*Risk Artifacts Filter*), que realiza a filtragem de métodos, classes e pacotes com base nos valores referência propostos. Serão também descritos, nos capítulos seguintes, o experimento (Capítulo 7) e os estudos de casos realizados (Capítulos 8 e 9) com o objetivo de avaliar os valores referência propostos.

[2]http://metrics2.sourceforge.net/

Capítulo 6

RAFTool - Ferramenta de Filtragem de Métodos, Classes e Pacotes

Neste capítulo é apresentada a ferramenta *RAFTool* (*Risk Artifacts Filter Tool)*, cujo propósito é realizar a filtragem de métodos, classes e pacotes que possuam 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 [Sommerville, 2010].

Um processo de medição que pode fazer parte de um processo de controle de qualidade de software é exibido na Figura 6.1 [Sommerville, 2010]. Um dos estágios chaves nesse processo é a identificação de medições anômalas, que é feita por meio da comparação das medidas obtidas com um histórico de medições coletadas de projetos anteriores, com o objetivo de identificar valores que estejam fora dos limites definidos como desejáveis. Os valores referência propostos neste trabalho de dissertação são capazes de indicar o que são esses valores "fora dos limites definidos como desejáveis" para as métricas estudadas. Isso facilita a aplicação desse processo de controle, afinal, não é prática comum das organizações a manutenção de um banco de dados de medições coletadas em projetos anteriores, como idealiza Sommerville [2010].

Além de a definição de valores referência ser de suma importância para a efetiva aplicação das métricas na indústria de software, o fornecimento de ferramentas apropriadas de medição e análise quantitativa de software é fundamental, pois de acordo com Sommerville [2010], o processo de medição deve ser simples e fácil.

Para a medição de softwares orientados por objetos, existem ferramentas que têm sido utilizadas em pesquisas relacionadas a métricas de softwares, tais como *Eclipse*

Metrics Plugin, *Google CodePro AnalytiX*[1] e *Moose Platform*[2]. Essas ferramentas realizam a medição de métricas de software, porém, não fornecem meios para realizar uma análise quantitativa dos resultados da medição, o que possibilitaria ao arquiteto de software a definição de políticas de restrição quantitativa para realizar a filtragem de artefatos que devem ser inspecionados.

Nesse contexto, como parte dessa dissertação, foi desenvolvida a ferramenta *RAFTool*, que suporta, baseada em uma entrada de medições de software em formato XML, a realização da filtragem de métodos, classes e pacotes de acordo com os valores referência propostos. Por exemplo, suponha que um arquiteto de software estabeleça, em sua política de qualidade de software, que os métodos avaliados por meio das métricas MLOC e NBD, nas faixas *Regular/Ocasional* e *Ruim/Raro*, respectivamente, devam ser inspecionados. *RAFTool* proporciona a construção dessa política de filtragem, baseada em expressões booleanas que traduzem as faixas estabelecidas para os valores referência, retornando os métodos que estão na condição estabelecida. Caso a quantidade de métodos retornados seja muito grande, tornando a inspeção manual inviável para o tempo disponibilizado pela organização à qualidade interna do software, o arquiteto pode tornar essa política mais restrita, adicionando outra restrição, como a classificação *Ruim/Raro* na métrica PAR, ou até mesmo aumentar o nível de restrição da métrica MLOC para a faixa *Ruim/Raro* do valor referência.

Este capítulo está organizado da seguinte forma: Seção 6.1 apresenta os requisitos funcionais de *RAFTool* por meio da descrição sucinta dos fluxos, entradas e saídas do sistema. Seção 6.2 mostra a arquitetura utilizada no desenvolvimento de *RAFTool*. Seção 6.3 exemplifica a utilização da ferramenta, com imagens da execução e saída, baseada nas métricas coletadas do código-fonte de uma versão inicial da própria ferramenta. Seção 6.4 conclui o capítulo, analisando as aplicações da ferramenta e extensões futuras.

6.1 Requisitos

Os requisitos funcionais de *RAFTool* se dividem em três grupos, que se referem à instanciação do sistema analisado, ao processo de filtragem e a visualização/exportação dos resultados. Os requisitos são descritos detalhadamente a seguir.

1. **Instanciação do Sistema Analisado**

[1] `https://developers.google.com/java-dev-tools/codepro/doc/`
[2] `http://www.moosetechnology.org/`

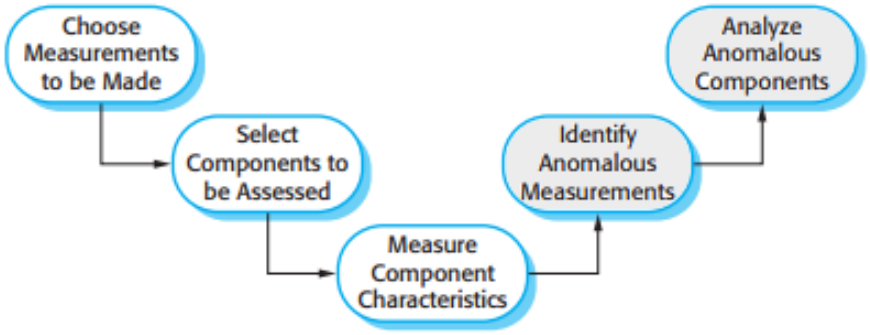


Figura 6.1: O processo de medição de produto. [Sommerville, 2010].

a) **Seleção do nome do sistema analisado.** *RAFTool* deve permitir a instanciação de um sistema a ser analisado a partir da digitação de um nome.

b) **Importação de medições por meio de arquivos XML do sistema analisado.** *RAFTool* deve permitir a importação de um ou mais arquivos XML que contenham medições de métricas de software suportadas. É necessário que a ferramenta importe mais de um arquivo XML para um único sistema pois ele pode ser constituído de vários projetos. O formato do arquivo XML de entrada é o mesmo formato da saída exportada pelo *Eclipse Metrics Plugin*. Isso ocorre porque o *Eclipse Metrics Plugin* foi a ferramenta utilizada para extrair o *dataset* de métricas usado no processo de derivação dos valores referência, além de ser uma ferramenta integrada com a IDE mais utilizada no mundo para desenvolvimento em Java, o Eclipse. Dessa forma, diferenças de medições de métricas de software que ocorrem por implementações distintas entre as ferramentas de extração e que geram ruído quando da utilização dos valores referência e a sua interpretação dos valores extraídos são evitadas. O ideal é a utilização conjunta do *Eclipse Metrics Plugin* e da *RAFTool*, apesar de não ser um fato obrigatório.

c) **Conclusão da instanciação do sistema analisado.** *RAFTool* deve permitir, após a seleção do nome e a importação das medições, a opção de concluir aquela instanciação do sistema analisado. Nesse momento, a ferramenta deve processar os arquivos XML, inserindo as medições em um banco de dados *standalone*, que persistirá os dados para trabalhos de filtragem até o término daquela execução do programa. Esses dados não ficam persistidos para uma próxima execução.

2. **Processo de Filtragem de Métodos, Classes ou Pacotes**

a) **Seleção do tipo de artefato filtrado.** O usuário deve selecionar o tipo de artefato filtrado: método, classe ou pacote. Não é possível filtrar mais de um tipo de artefato de forma simultânea.

b) **Criação da expressão booleana de filtragem de artefatos.** O usuário deve digitar a expressão booleana de filtragem de métodos, classes ou pacotes. Essa expressão booleana deve ser uma combinação de *ANDs* e *ORs* dentro das faixas dos valores referência das métricas que serão utilizadas na filtragem que está sendo criada. Para isso, o seguinte formato é definido: *COMMON*|*CASUAL*|*UNCOMMON*[*METRIC_ID*]. *COMMON* corresponde à faixa *Bom/Regular*, *CASUAL* à faixa *Regular/Ocasional* e *UNCOMMON* à faixa *Ruim/Raro*. Quando o usuário escolher a faixa *CASUAL*, o sistema deverá retornar as métricas, classes e pacotes que estejam na faixa *Regular/Ocasional* ou *Ruim/Raro*. *METRIC_ID* é o identificador da métrica que deve estar naquela faixa, tal como exibido na Tabela 4.2. Logo, se o usuário digitar *CASUAL*[*MLOC*], por exemplo, o processo de filtragem deve retornar todos os métodos do sistema instanciado cuja classificação pelo valor referência sugerido para a métrica Número de Linhas de Código por Método for *Regular/Ocasional*. Com a utilização de *ANDs* e *ORs*, o usuário poderá compor sua filtragem com vários valores referência. Por exemplo, se o usuário digitar *CASUAL*[*MLOC*] *AND UNCOMMON*[*NBD*], o processo de filtragem deve retornar todos os métodos do sistema instanciado classificados como *Regular/Ocasional* para a métrica Número de Linhas de Código por Método e, desses métodos, todos aqueles classificados como *Ruim/Raro* pela métrica Profundidade de Blocos Aninhados. Parênteses devem ser aceitos nessa expressão como forma de indicação da precedência das operações.

c) **Ordenação dos artefatos retornados.** O usuário também pode escolher a métrica a ser utilizada como critério de ordenação descendente dos artefatos retornados. Caso opte por não escolher, a exibição dos resultados se dará de forma aleatória.

d) **Execução da filtragem.** *RAFTool* deve executar a filtragem conforme os parâmetros estabelecidos, acessando o banco de dados criado na fase de instanciação, buscando os artefatos e selecionando-os. O resultado da filtragem deve ser ordenado nessa execução.

3. **Visualização/Exportação dos Resultados**

a) **Visualização dos artefatos filtrados.** Os artefatos filtrados devem ser exibidos em uma tabela. Para o caso dos métodos, devem ser exibidos o nome do método, o nome da classe e o nome do pacote. Para as classes, devem ser exibidos o nome, o nome do arquivo da classe e o nome do pacote. A exibição do nome do arquivo é útil pois um arquivo pode conter mais de uma classe. Para os pacotes, são exibidos somente os nomes.

b) **Exportação dos artefatos filtrados para CSV.** Os artefatos filtrados podem ser exportados para uma planilha em formato CSV, arquivo escolhido pelo usuário na hora da exportação.

4. **Outros Requisitos**

a) **Fluxo entre janela de visualização e de filtragem.** Uma vez que a filtragem foi acionada, a janela de visualização deverá exibir os resultados. Quando essa janela for fechada, o usuário pode alterar os campos de filtro, executando quantos processo de filtragem forem necessários.

b) **Reinicialização do sistema instanciado.** Uma vez que o sistema foi instanciado, a filtragem e visualização/exportação dos resultados só poderão ser executadas sobre aquele sistema. Se for necessária a instanciação de um novo sistema, o usuário tem a opção de realizar uma reinicialização, o que lhe possibilita a nova instanciação.

6.2 Arquitetura

6.2.1 *Model-View-Controller* (MVC)

Para o desenvolvimento da ferramenta *RAFTool* foi utilizado o padrão arquitetural *Model-View-Controller* (MVC). De acordo com Sommerville [2010], o padrão MVC separa a apresentação e a interação com a lógica de negócio e dados do sistema. A camada de Modelo gerencia os dados do sistema e as operações sobre esses dados. A Visão define e gerencia como os dados são apresentados para o usuário. O Controle gerencia a interação do usuário e passa essas interações tanto para a camada de Visão como para a camada de Modelo. Figura 6.2 mostra o padrão arquitetural MVC. Nas Seções 6.2.1.1, 6.2.1.2 e 6.2.1.3 são descritos os detalhamentos de cada uma dessas camadas em relação ao contexto da *RAFTool*.

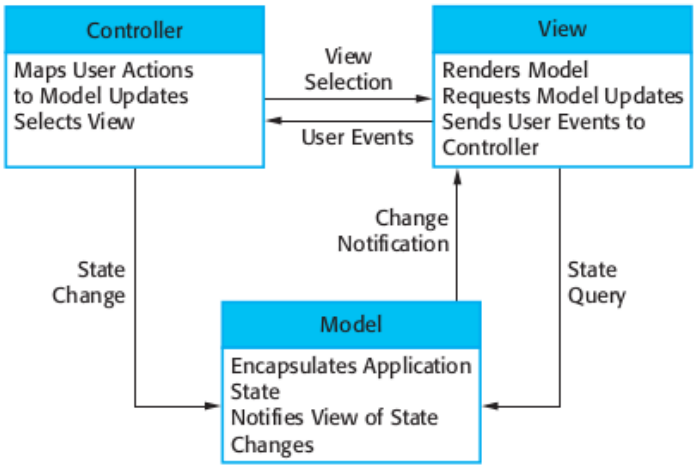


Figura 6.2: Padrão arquitetural MVC. Fonte: Sommerville [2010]

6.2.1.1 Modelo

O Modelo encapsula, além dos objetos de domínio, o acesso ao banco de dados utilizado pela aplicação. Outra responsabilidade do modelo dentro da *RAFTool* é relativo à lógica de negócio da aplicação, que são objetos cujas responsabilidades residem em realizar a lógica relativa ao processo de filtragem. Esse processo de filtragem envolve a interpretação das expressões booleanas utilizadas pelos usuários, bem como a sua aplicação nos artefatos selecionados. Utilizou-se o *framework* Javaluator[3], que auxilia na avaliação de expressões regulares na plataforma Java.

Os objetos que realizam o processo de filtragem precisam acessar os dados persistentes, o que é feito por meio do padrão de projeto *Data Access Object*[4]. Esse padrão encapsula em objetos do tipo DAO a complexidade de acesso a banco de dados, provendo uma interface de acesso aos dados persistentes que retornam para os solicitantes os objetos de domínio já construídos.

Na parte de persistência, foi utilizado o banco de dados HSQLDB[5]. HSQLDB é um servidor de banco de dados desenvolvido na plataforma Java e de código aberto. É uma solução simples de persistência de dados e que atende de forma plena os requisitos do projeto. Não requer pré-instalações na máquina em que a aplicação será executada, estando totalmente embarcado na aplicação. Para acesso ao banco de dados, foi utilizado o *driver* JDBC (*Java Database Connectivity*) para conexão com o banco de dados HSQLD. O *driver* JDBC fornece um conjunto de classes e interfaces que realizam a comunicação com o banco de dados.

[3] `http://javaluator.sourceforge.net/en/home/`
[4] `http://www.oracle.com/technetwork/java/dataaccessobject-138824.html`
[5] `http://hsqldb.org/`

6.2.1.2 Visão

A Visão foi desenvolvida com a utilização do *Swing*[6]. Swing implementa um conjunto de componentes para construir interfaces gráficas para interação com usuários (GUIs). Os componentes *Swing* são implementados diretamente em Java. Utilizou-se a IDE Netbeans[7], que possui recursos de construção de interfaces *Swing* com a utilização de *Drag and Drop*, facilitando a construção de interfaces gráficas.

6.2.1.3 Controle

O controle tem a responsabilidade de mapear as ações dos usuários em ações da camada de Modelo. Além disso, mediante a resposta da Camada de Modelo, ele é responsável por selecionar a visão apropriada ao usuário. Se a interface for trocada, por exemplo, o Controle cuida para que a camada de Modelo se mantenha inalterada.

6.2.2 Estrutura de Pacotes do Sistema

Nesta seção são documentados os pacotes do sistema, suas responsabilidades e como interagem entre si. A Figura 6.3 mostra o diagrama de pacotes de *RAFTool*, que exibe os módulos do sistema e as dependências entre eles, ilustrando a arquitetura proposta para o sistema.

1. *br.dcc.ufmg.view*: agrupamento das classes responsáveis pela camada de visão do sistema. Possui dependência com o módulo *controller*.

2. *br.dcc.ufmg.controller*: agrupamento das classes responsáveis pela camada de controle do sistema. Possui dependência com o a *view*, bem como com os restante dos módulos do sistema, que constituem a camada de modelo.

3. *br.dcc.ufmg.xml*: agrupamento das classes que trabalham com a leitura dos documentos XML por meio do *framework* JAXB, conversão para objetos Java e importação dos dados lidos desses documentos para o banco de dados. Para tal, possui dependência em relação ao módulo *persistence*, responsável pelas operações de banco de dados.

4. *br.dcc.ufmg.filtering*: agrupamento de classes responsáveis pelas operações de filtragem a serem realizadas nos artefatos do software analisado. Esse módulo do

[6] `http://docs.oracle.com/javase/7/docs/technotes/guides/swing/`
[7] `https://netbeans.org/`

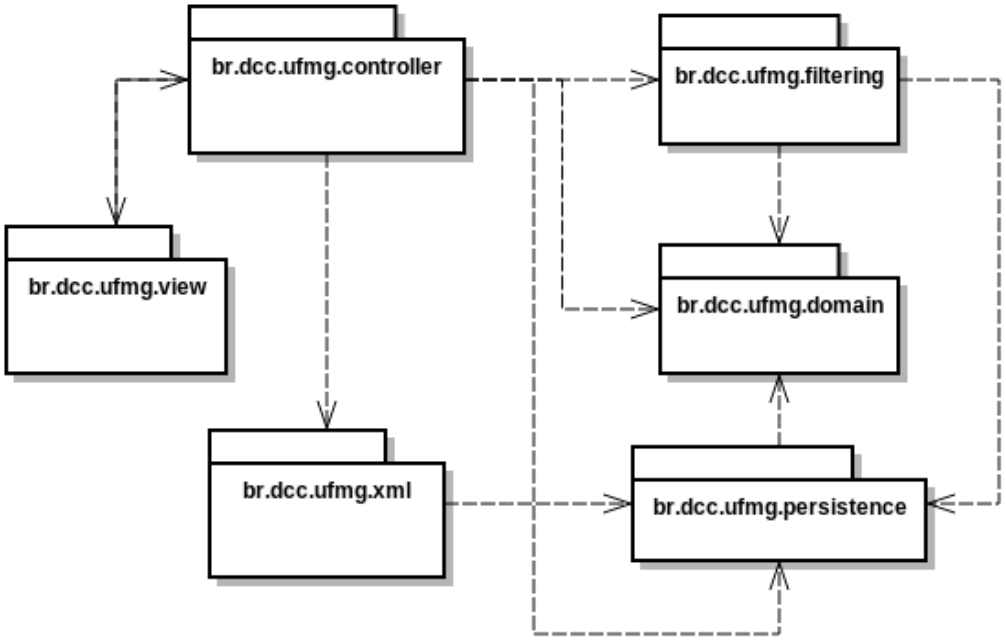


Figura 6.3: Diagrama de pacotes da *RAFTool*.

sistema possui dependência com o módulo *persistence*, que é responsável por acessar os dados do sistema analisado. Também possui dependência com o módulo *domain*, que modela o cenário real de um software orientado por objetos. É importante notar que esse módulo é independente do módulo *xml*, utilizando dados a nível de domínio.

5. *br.dcc.ufmg.domain*: agrupamento de classes responsáveis por representar o vocabulário e descrever os conceitos chaves de um software orientado por objetos, bem como as relações entre as entidades estabelecidas. Sobre os objetos do domínio são executadas as operações de filtragem de dados necessária ao funcionamento do software.

6. *br.dcc.ufmg.persistence*: agrupamento de classes responsáveis por tratar operações de inserção, leitura e atualização dos dados utilizados no sistema. É um módulo independente, que modela o domínio do problema.

6.2.3 Principais Classes do Sistema

Nesta seção são documentadas as principais classes do sistema, que são:

1. *br.dcc.ufmg.domain.Threshold*: estrutura de dados que agrupa os valores referência por meio de um conjunto fixo de constantes, funcionando como uma lista de

valores pré-definidos.

2. *br.dcc.ufmg.view.FilterJFrame*: classe responsável por desenhar a tela principal do sistema, na qual ocorre a entrada de dados para o processo de filtragem, bem como o tratamento dos eventos da mesma.

3. *br.dcc.ufmg.view.FilterResultJFrame*: classe responsável por desenhar a tela do sistema na qual é exibida o resultado da filtragem. Essa tela é acionada por meio da classe *br.dcc.ufmg.view.FilterJFrame*.

4. *br.dcc.ufmg.controller.FilterController*: classe responsável por tratar as ações do usuário na tela principal do sistema, acionando outros componentes do sistema em razão dessa interação. Funciona como mediadora entre a visão e modelo. Essa classe também tem a responsabilidade de montar a visão apropriada ao usuário quando são exibidos os artefatos filtrados, mediando o resultado do domínio em relação a visão.

5. *br.dcc.ufmg.controller.FilterResultController*: classe responsável por tratar as ações do usuário na tela do sistema na qual é exibida o resultado da filtragem. Essa classe também é responsável por selecionar a visão apropriada em resposta a ações do usuário.

6. *br.dcc.ufmg.persistence.ArtifactDAO*: classe abstrata que implementa um modelo de classe que tem por objetivo realizar o acesso aos dados referentes aos artefatos de software importados no sistema. Essa classe é herdada pelas implementações concretas nas classes *PackagesDAO*, *ClassesDAO* e *MethodsDAO*, delegando a elas responsabilidades mínimas referentes a suas especificidades.

7. *br.dcc.ufmg.persistence.MetricDAO*: classe que implementa o acesso aos dados das medidas importadas para o sistema analisado, bem como as operações de inserção desses dados lidos dos arquivos XML no banco de dados.

8. *br.dcc.ufmg.persistence.HSQLDBConnection*: implementa o padrão de projeto *Singleton* que garante a existência de apenas uma conexão com o banco de dados criado. Essa classe encapsula detalhes de implementação da conexão com o banco de dados instanciado.

9. *br.dcc.ufmg.xml.ParserXML*: implementa a leitura dos documentos XML que contém as medidas das métricas para o sistema analisado, utilizando o *framework* JAXB, que converte a árvore do documento XML para objetos Java.

10. *br.dcc.ufmg.xml.XmlFileToDatabase*: implementa uma interface de comunicação que aciona o módulo de leitura dos dados dos documentos XML (ParserXML) e aciona o objeto *MetricDAO* que insere as medições no banco de dados.

11. *br.dcc.ufmg.filtering.ArtifactFilter*: classe abstrata que implementa um modelo de classe que tem por objetivo realizar o processo de filtragem dos dados referentes aos artefatos de software. Essa classe é herdada pelas implementações concretas nas classes *PackagesFilter*, *ClassesFilter* e *MethodsFilter*, delegando a elas responsabilidades mínimas referentes a suas especificidades.

12. *br.dcc.ufmg.filtering.MeasuresEvaluator*: essa classe implementa as operações de avaliação do método, classe ou pacote em relação à expressão booleana de filtragem especificada pelo usuário. Para cada método, classe ou pacote do sistema instanciado, é criada uma instância de *MeasuresEvaluator*, que recebe as medições referentes àquele artefato. Posteriormente, baseada na expressão booleana, avalia se o artefato atende os critérios estabelecidos. Essa classe é uma filha da classe *AbstractEvaluator*, que faz parte do *framework* Javaluator. Sua especificação define como será a "tradução" dos termos que compõem a expressão para *true* ou *false*. A operação sobrescrita da classe mãe *getValue* recebe o trecho literal que é parte da expressão, por exemplo, $CASUAL[MLOC]$ e, dada as medidas de métricas adicionadas na instância e os limites estabelecidos no *enum Thresholds*, o interpreta como *true* ou *false*. A combinação geral dos termos da expressão é conduzida pelo *framework*.

13. *br.dcc.ufmg.domain.Class*: encapsula atributos e comportamentos referentes a abstração do tipo classe, que representa uma classe de um sistema orientado por objetos. A partir da leitura dos documentos XML do sistema avaliado, são criadas instâncias dessa classe que correspondem às classes medidas do sistema.

14. *br.dcc.ufmg.domain.Method*: encapsula atributos e comportamentos referentes a abstração do tipo método, que representa um método de uma classe (objeto *Class*) de um sistema orientado por objetos. A partir da leitura dos documentos XML do sistema avaliado, são criadas instâncias dessa classe que correspondem aos métodos medidos de uma determinada classe do sistema.

15. *br.dcc.ufmg.domain.Package*: encapsula atributos e comportamentos referentes a abstração do tipo pacote, que representa um pacote de um sistema orientado por objetos. A partir da leitura dos documentos XML do sistema avaliado, são criadas instâncias dessa classe que correspondem aos pacotes medidos do sistema.

16. *br.dcc.ufmg.domain.Measure*: encapsula atributos e comportamentos referentes a abstração do tipo medição, que representa a ação de medir um conjunto de métricas para um determinado artefato. As classes *Class*, *Method* e *Package* herdam de *Measurement*, pois é uma característica comum dessas entidades.

6.3 Utilização

Para exemplificar a utilização da ferramenta *RAFTool*, foi extraído, via *Eclipse Metrics Plugin*, o arquivo XML de medições do código-fonte da própria ferramenta *RAFTool*. Os passos desse exemplo são enumerados a seguir:

1. Selecionar o nome do sistema analisado, no caso, *RAFTool*. Isso é feito por meio da digitação do nome do sistema no campo *System Name*, conforme ilustrado na Figura 6.4.

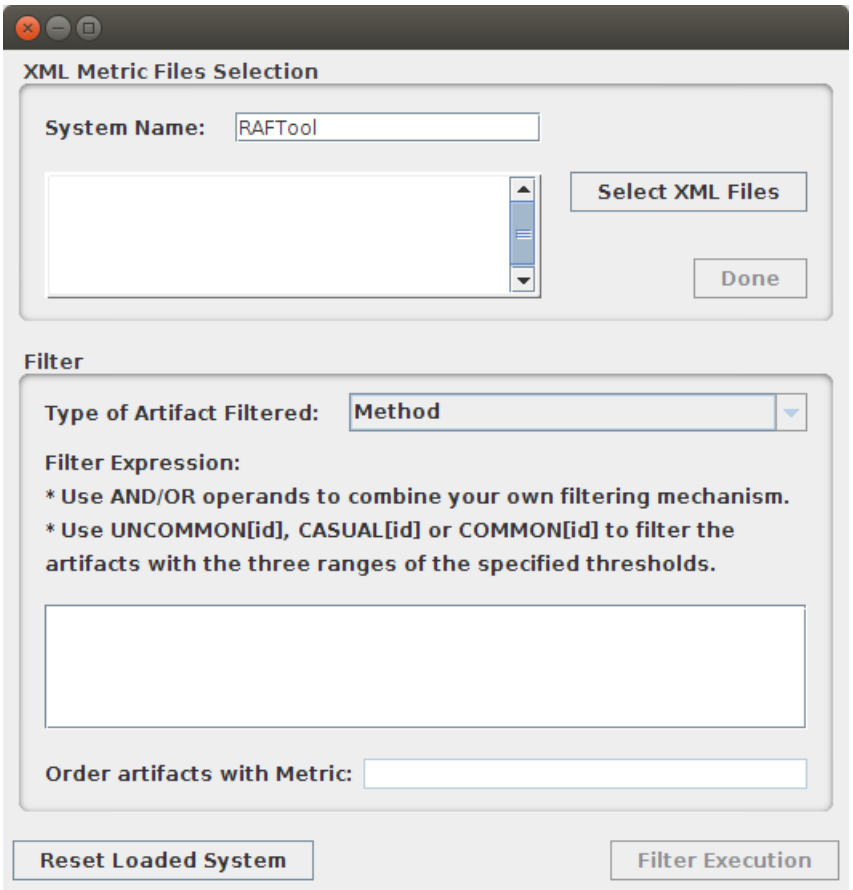


Figura 6.4: Instanciação de sistema analisado: seleção do nome do sistema analisado.

2. Importar o arquivo XML extraído via *Eclipse Metrics Plugin* para o sistema que está sendo instanciado. Isso é feito por meio da opção *Select XML Files*, na qual

o usuário pode selecionar um ou mais arquivos XML que contenham medições referentes ao sistema. No exemplo, somente um arquivo foi selecionado, *risk.xml*, como pode ser visto na Figura 6.5.

3. Concluir a instanciação do sistema analisado, por meio do botão *Done* (Figura 6.5). Quando isso ocorre, os dados referentes às medições são importados no HSQLDB e os campos referentes a instanciação são desabilitados pois não podem mais sofrer alterações.

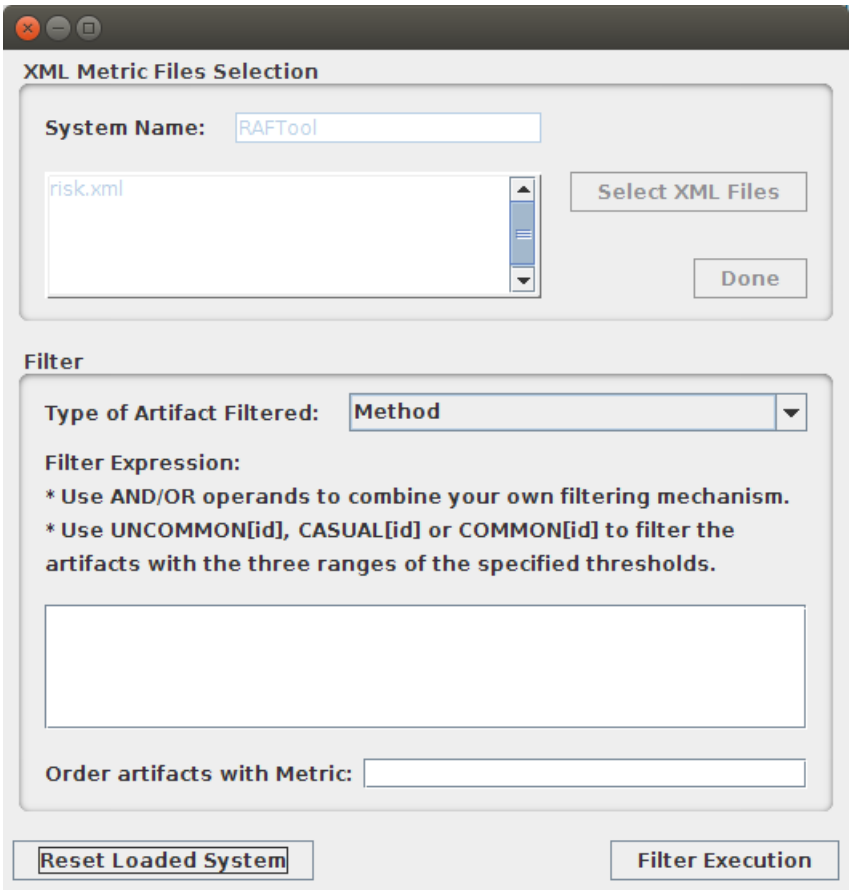


Figura 6.5: Instanciação de sistema analisado.

4. Posteriormente, o usuário deve selecionar o tipo de artefato filtrado, no campo *Type of Artifact Filtered*. As opções são *Method*, *Class* ou *Package*. No exemplo, deve-se selecionar a opção *Method*, como pode ser visto na Figura 6.5.

5. O próximo passo consiste na criação da expressão booleana de filtragem de artefatos, que deve ser digitada no campo *Filter Expression*. No exemplo, foi utilizada a expressão $CASUAL[MLOC]$ (Figura 6.6).

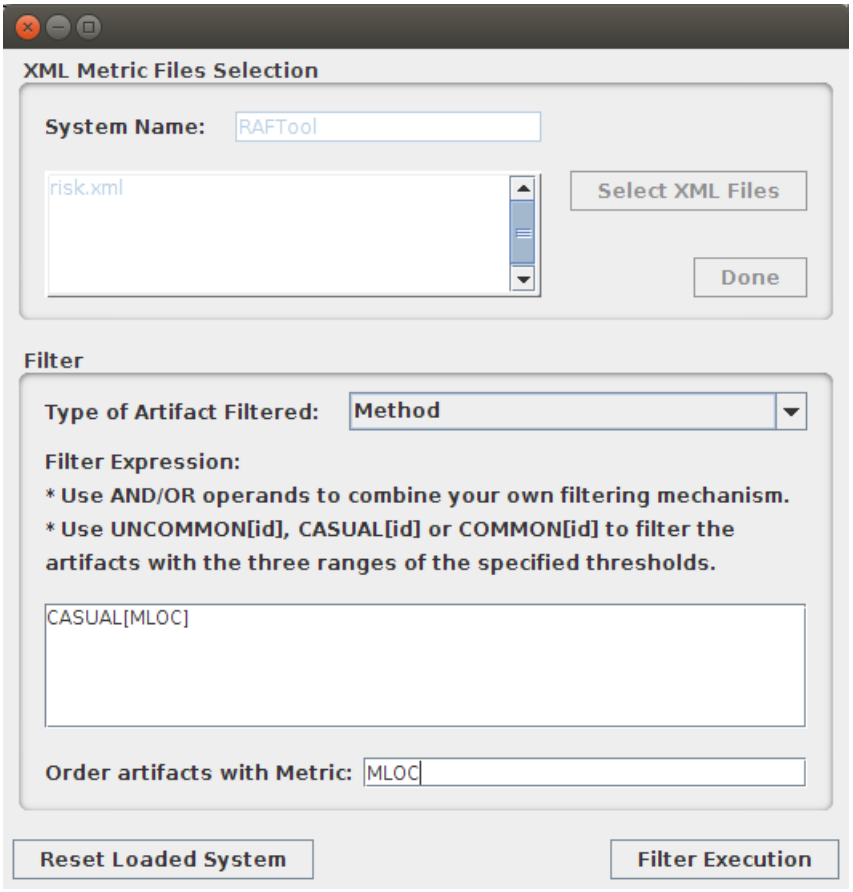


Figura 6.6: Processo de filtragem de métodos, classes ou pacotes: criação da expressão booleana de filtragem de artefatos e estabelecimento do critério de ordenação.

6. Feito isso, o usuário deve escolher uma métrica para ser o critério de ordenação descendente dos artefatos retornados, por meio do campo *Order artifacts with metric*. No exemplo, escolhemos a métrica MLOC (Figura 6.6). Com os campos preenchidos, o usuário pode acionar a filtragem por meio da opção *Filter Execution*.

7. Após a opção *Filter Execution* ser acionada, *RAFTool* abre uma nova janela com a visualização dos resultados, conforme observado na Figura 6.7. Nessa tela são exibidos a quantidade de resultados retornados, no caso, 13, e os campos dos métodos filtrados (*Name*, *Class* e *Package*) são exibidos por meio de uma tabela.

8. Se o usuário refinar a expressão booleana de filtragem, ela retornará menos métodos, como pode ser visto na Figura 6.8. Nesse caso, a expressão booleana utilizada foi $CASUAL[MLOC]$ AND $CASUAL[VG]$, que retornou 6 métodos.

9. Na tela de visualização (Figuras 6.7 e 6.8) há a opção *Export Results in CVS*

Format, que proporciona ao usuário a opção de gerar um arquivo CSV com a tabela de resultados.

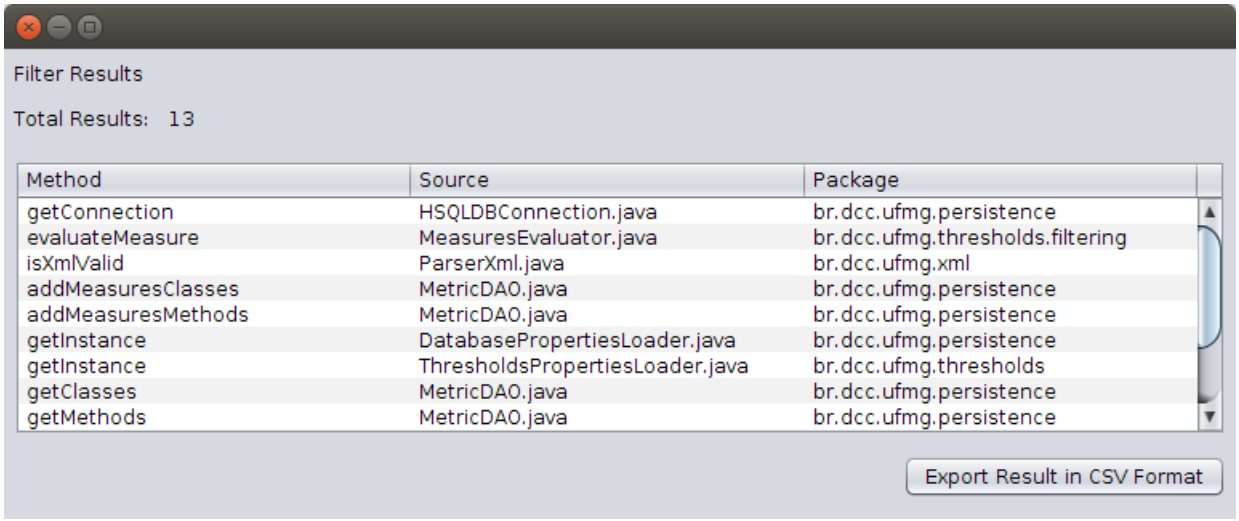


Figura 6.7: Visualização dos resultados: $CASUAL[MLOC]$

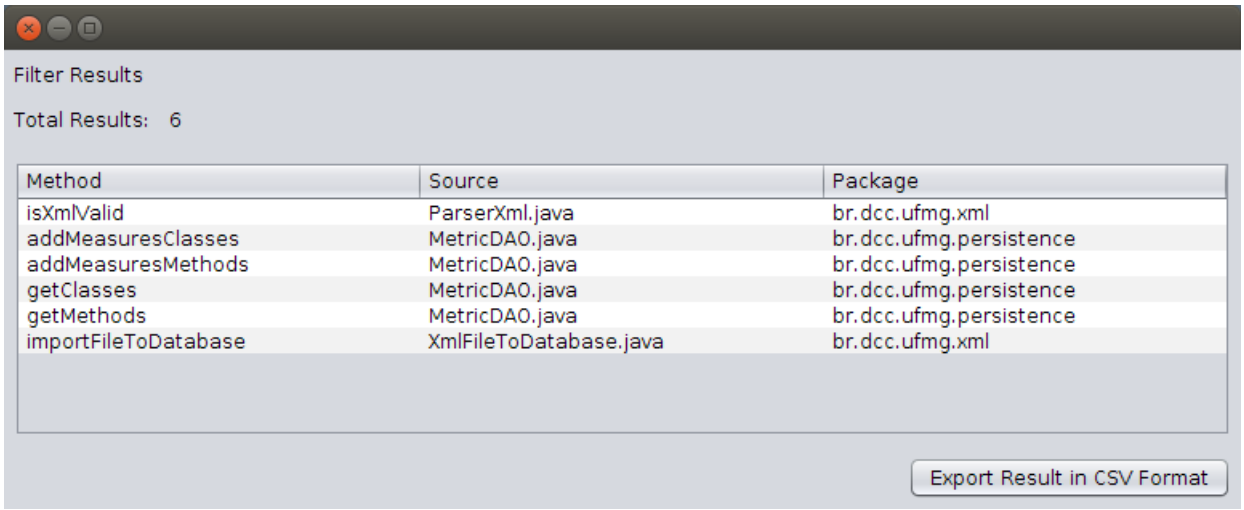


Figura 6.8: Visualização dos resultados: $CASUAL[MLOC]$ *AND* $CASUAL[VG]$

6.4 Conclusão

A ferramenta *RAFTool* realiza a filtragem de métodos, classes e pacotes que possuem medições anômalas de métricas de softwares orientados por objetos segundo os valores referência sugeridos neste trabalho de dissertação. Como exemplo da utilização da *RAFTool*, disponibilizamos um vídeo-tutorial *online* que mostra o uso da ferramenta:

`http://www.dcc.ufmg.br/~tfilo/raftool/demo_raftool.htm`

Para instalar e utilizar a *RAFTool*, é necessário que a máquina virtual Java esteja instalada, com versão igual ou superior a 1.6. O executável está disponível em:

`http://www.dcc.ufmg.br/~tfilo/raftool/raftool.jar`

RAFTool complementa o trabalho de identificação de valores referência realizado nesta dissertação, fornecendo uma ferramenta que suporta a utilização desses padrões de forma efetiva na filtragem de métodos, classes e pacotes com medições anômalas. Dessa forma, espera-se que, em conjunto com o *Eclipse Metrics Plugin* e os valores referência sugeridos nesse trabalho de dissertação, *RAFTool* colabore com os esforços de pesquisa que visam a aplicar de forma efetiva as métricas de software no contexto do gerenciamento da qualidade interna de software por meios quantitativos.

Como trabalhos futuros, vislumbramos a integração da *RAFTool* ao *plugin Eclipse Metrics*, monitorando o código de forma dinâmica durante o desenvolvimento, manutenção e evolução de software, disparando alarmes que possibilitem aos desenvolvedores a utilização de métricas de forma totalmente integrada à ferramenta de desenvolvimento. Nessa ferramenta o desenvolvedor não precisaria se preocupar com números, pois os valores referência propostos funcionariam como sinais, fazendo uma analogia com os sinais de trânsito. A faixa *Bom/Frequente* seria o sinal "verde", a faixa *Regular/Ocasional* o sinal "amarelo", sugerindo atenção, e a faixa *Ruim/Raro* o sinal "vermelho", indicando que deve-se parar e analisar o que está sendo codificado.

O código-fonte da *RAFTool* está disponível no *GoogleCode*[8], que hospeda projetos em um ambiente colaborativo de desenvolvimento. Utilizou-se o *Subversion*[9] como ferramenta de controle de versão. O projeto pode ser baixado diretamente no *Eclipse* e está disponível por meio do seguinte link:

`https://code.google.com/p/raftool/source/checkout`

A ferramenta *RAFTool* foi utilizada na condução dos Estudos de Casos 1 e 3, que serão relatados nos Capítulo 8 e 9, respectivamente.

[8] `https://code.google.com/`
[9] `http://subversion.apache.org/`

Capítulo 7

Avaliação dos Valores Referências como Indicativo de Qualidade

Este capítulo apresenta um experimento realizado com o objetivo de avaliar valores referência propostos para as métricas de pacote como indicativo de qualidade em processos de reestruturação de softwares orientados por objetos. Esse experimento foi inspirado no trabalho de Anquetil & Laval [2011], que avaliaram métricas arquiteturais de qualidade de software em ações reais de reestruturação, observando os valores das métricas antes e depois da reestruturação. Métricas adequadas devem ser capazes de medir o acréscimo da qualidade de software. Da mesma forma, os valores referência sugeridos devem ser capazes de medir o acréscimo da qualidade de software de forma quantitativa.

7.1 Experimento

Nesta seção são descritos os processos que envolvem a preparação e condução do experimento. O escopo do experimento é definido como: analisar valores referência propostos para um conjunto de métricas de softwares orientados por objetos com o objetivo de avaliá-los em relação à sua capacidade de identificar o acréscimo de qualidade de software do ponto de vista de pesquisadores no contexto de processos reais de reestruturação de pacotes em softwares orientados por objetos. Seção 7.1.1 define o contexto do experimento. Seção 7.1.2 trata a seleção das variáveis e formulação das hipóteses relacionadas ao experimento. Seção 7.1.3 define o que está sendo considerado como reestruturação.

7.1.1 Seleção do Contexto

O contexto do experimento são pacotes de softwares orientados por objeto que tenham sido reestruturados. Foram definidas duas situações ideais referentes às amostras para o experimento proposto:

- o processo de reestruturação deve ser um esforço "puro" e de duração limitada [Anquetil & Laval, 2011]. Isso permite que se possa identificar as versões antes e depois da reestruturação, de forma que elas não se percam em outras modificações naturais da evolução do software. Isso é difícil de se encontrar, pois sistemas reais precisam evoluir;

- os valores referência sugeridos foram derivados a partir de medições coletadas com o plugin *Metrics*[1] para o *Eclipse*[2], desenvolvidos na plataforma *Java*. Isto impõe uma restrição: os projetos devem ter condições de ser compilados no *Eclipse*, de forma que os valores coletados para o experimento nos sistemas alvo sejam coletados da mesma forma que os dados que originaram os valores referência.

Foram utilizados dois softwares bem conhecidos que atendem a esses requisitos: *JHotDraw* e *Eclipse*. Na Seção 7.2 esses sistemas são descritos em detalhes, bem como os processos de reestruturação considerados.

7.1.2 Seleção da Variável e Formulação das Hipóteses

A *variável independente* para o experimento é a versão considerada do sistema, ou seja, a versão anterior (versão base) ou a versão posterior (versão reestruturada). A *variável dependente* representa a faixa em que o pacote avaliado foi classificado pelo valor referência sugerido. Dessa variável dependente, pode-se derivar a porcentagem de pacotes relacionados à reestruturação classificados em cada uma das faixas sugeridas pelo valor referência. Formalizam-se, então, as seguintes hipóteses, para o valor referência identificado para a métrica m:

- H_m^{null}: a porcentagem de pacotes avaliados pela métrica m na faixa *Ruim/Raro* em favor de pacotes avaliados na faixa *Regular/Ocasional* e *Bom/Frequente*, bem como a porcentagem de pacotes avaliados pela métrica m na faixa *Regular/Ocasional* em favor de pacotes avaliados na faixa *Bom/Frequente*, é maior ou igual

[1] `http://metrics2.sourceforge.net/`
[2] `http://www.eclipse.org/`

na versão reestruturada do que na versão base correspondente, no contexto da reestruturação[3].

- H_m^a: a porcentagem de pacotes avaliados pela métrica m na faixa *Ruim/Raro* em favor de pacotes avaliados na faixa *Regular/Ocasional* e *Bom/Frequente*, bem como a porcentagem de pacotes avaliados pela métrica m na faixa *Regular/Ocasional* em favor de pacotes avaliados na faixa *Bom/Frequente*, é menor na versão reestruturada do que na versão base correspondente, no contexto da reestruturação.

Foi utilizada a porcentagem porque, em termos absolutos, a quantidade de pacotes na faixa *Ruim/Raro* pode aumentar na versão reestruturada. Porém, em termos relativos, espera-se que ela diminua. Por exemplo, um pacote classificado como *Ruim/Raro* pode ser decomposto em 10 pacotes, sendo que 2 deles são *Ruim/Raro* e o restante *Bom/Frequente*. Nesse caso, ocorreu um aumento absoluto de pacotes *Ruim/Raro* na versão reestruturada, porém, houve um considerável decréscimo relativo, de 100% para 20%. Como busca-se avaliar esforços puros de reestruturação, ou seja, aqueles em que não há implementação de novas funcionalidades ou melhorias, nessa situação, em termos globais, a reestruturação aumentou a qualidade dos pacotes do software e o valor referência sugerido é capaz de indicar isso. Portanto, não se tem por premissa que a reestruturação resultará, necessariamente, somente em pacotes classificados como *Bom/Frequente* ou *Regular/Ocasional*, mas espera-se que a qualidade global da porção reestruturada do sistema aumente e que os valores referência sejam capazes de medir esse acréscimo.

7.1.3 Reestruturação

Na avaliação foram conduzidos testes em relação a reestruturação global e local. Reestruturação global é aquela que afeta o sistema por inteiro. Nesse caso, foram avaliadas as hipóteses considerando todos os pacotes do sistema. A reestruturação global deve melhorar a qualidade do sistema como um todo [Anquetil & Laval, 2011].

A reestruturação local não pode considerar todos os pacotes do sistema, afinal, apesar do aumento da qualidade dos pacotes relacionados a reestruturação, a qualidade do sistema como um todo pode ser deteriorada em decorrência de outras modificações. Nesse caso, foram considerados apenas os pacotes relacionados ao contexto da reestruturação nas versões base e reestruturada, ou seja, somente a parte reestruturada

[3]Sugere-se que não há relação entre a reestruturação e o acréscimo de qualidade indicado pelo valor referência

Figura 7.1: Preparação e Operação do Experimento

do sistema. A reestruturação local envolve as seguintes possibilidades relacionadas aos pacotes:

- **Novo:** se um pacote é introduzido na versão reestruturada para agrupar classes que já existiam na versão base, ele é considerado como parte do contexto da reestruturação, bem como os pacotes fontes das classes reagrupadas.

- **Transferência de Classes:** se uma classe é transferida de um pacote para outro, considera-se que ambos os pacotes melhoraram sua qualidade [Anquetil & Laval, 2011]. Pressupõe-se que a reestruturação pode ser realizada movendo as classes entre os pacotes e, quando isso ocorre, espera-se que classe tenha sido transferida para um pacote com o qual tenha maior relação. Por tal motivo, supôs-se que a classe não estava bem colocada, e, portanto, o pacote do qual a classe foi movida melhorou sua qualidade. De forma similar, supôs-se que a classe está mais bem colocada após a reestruturação, e, portanto, o pacote para o qual a classe foi movida também melhorou sua qualidade. É importante notar que não se considera que os pacotes são o melhor agrupamento possível para o projeto, mas apenas que eles tiveram sua qualidade aumentada.

7.1.4 Operação

Após definidos o escopo e o planejamento do experimento, é necessário preparar os sistemas para o experimento. Para tal, foi feito o *download* dos projetos identificados na seleção do contexto e a importação deles no *Eclipse*, previamente instalado com o *plugin Metrics*. O experimento foi realizado por meio da execução do *plugin Metrics* nos projetos, coletando os valores das métricas de pacote cujos valores referência serão avaliados. Esses dados foram importados em um banco de dados *Mysql* com o intuito de facilitar a sua análise e validação. A Figura 7.1 sumariza a operação do experimento.

7.2 Sistemas Alvo

Nesta seção descrevemos as reestruturações identificadas nos sistemas alvo.

Tabela 7.1: Reestruturações locais - JHotDraw

JHotDraw	**7.3.1/7.4.1**	**7.4.1/7.5.1**	**7.5.1/7.6**
Pacotes versão base	46	62	65
Pacotes Removidos	2	0	1
Pacotes Adicionados	16	3	1
Pacotes Reestruturados	2	3	1
Pacotes Reestruturados[5]	18	4	2
Pacotes versão reestruturada	62	65	65

7.2.1 *JHotdraw*

JHotdraw[4] é um *framework* desenvolvido em Java para desenho técnico e gráfico. Foi desenvolvido como um exercício de padrões de projeto em suas primeiras versões, mas é uma ferramenta gráfica poderosa. Foram identificadas, por meio do trabalho de Anquetil & Laval [2011], as versões que passaram por reestruturações. Via documentação das versões e com a análise do código fonte dos projetos, foram identificadas 3 mudanças de versões com reestruturações locais, objetos do experimento:

1. os pacotes *org.jhotdraw.draw* e *org.jhotdraw.app.action*, na Versão 7.4.1, tiveram parte de suas classes reagrupadas em 11 e 5 subpacotes, respectivamente. Tanto os pacotes reestruturados como os pacotes adicionados em decorrência da reestruturação fazem parte do contexto da reestruturação;

2. o pacote *org.jhotdraw.gui.plaf.palette*, na Versão 7.5.1, teve parte de suas classes reagrupadas em um novo subpacote; e classes do pacote *org.jhotdraw.gui.event* foram migradas para o pacote *org.jhotdraw.draw.event*;

3. o pacote *org.jhotdraw.gui.plaf.palette*, na Versão 7.6, teve parte de suas classes reagrupadas em um novo subpacote e algumas classes do pacote *org.jhotdraw.gui.event* foram migradas para o pacote *org.jhotdraw.draw.event*.

A Tabela 7.1 sumariza quantitativamente as reestruturações locais identificadas.

7.2.2 *Eclipse*

Eclipse[6] é uma *IDE* para desenvolvimento Java e outras linguagens de programação, sendo a *IDE* Java mais utilizada no mundo. *Eclipse* foi desenvolvida em Java e é

[4] `http://www.jhotdraw.org/`

[5] Inclui os pacotes reestruturados da versão base somados com os pacotes adicionados em decorrência da reestruturação, totalizando os pacotes relacionados à reestruturação na versão reestruturada. Fazem parte do total de pacotes na versão reestruturada.

[6] `http://www.eclipse.org`

open source. Foram identificadas, por meio do trabalho de Anquetil & Laval [2011], e via documentação[7] disponível, a reestruturação da Versão 2.1.3 para 3.0, na qual *Eclipse* evoluiu do conceito de uma *IDE* extensível para *RCP* (*Rich Client Platform*). A reestruturação considerada é global pois tem impacto em um grande número de pacotes [Anquetil & Laval, 2011].

7.3 Resultados

Nesta seção são apresentados os resultados obtidos no experimento conduzido com os softwares *JHotDraw* e *Eclipse*. Como o experimento envolve a restruturação de pacotes em softwares orientados por objetos, os resultados são reportados para cada uma das métricas de pacotes analisadas neste trabalho: Número de Classes (NOC), Acoplamento Aferente (CA), Acoplamento Eferente (CE) e Distância Normalizada (RMD). A Tabela 7.2 traz os valores referência identificados para as métricas usadas nesse experimento.

Tabela 7.2: Valores referência identificados para as métricas NOC, CA, CE e RMD.

Métrica	**Bom/Frequente**	**Regular/Ocasional**	**Ruim/Raro**
NOC	$NOC \leq 11$	$11 < NOC \leq 28$	$NOC > 28$
CA	$CA \leq 7$	$7 < CA \leq 39$	$CA > 39$
CE	$CE \leq 6$	$6 < CE \leq 16$	$CE > 16$
RMD	$RMD \leq 0,467$	$0,467 < RMD \leq 0,750$	$RMD > 0,750$

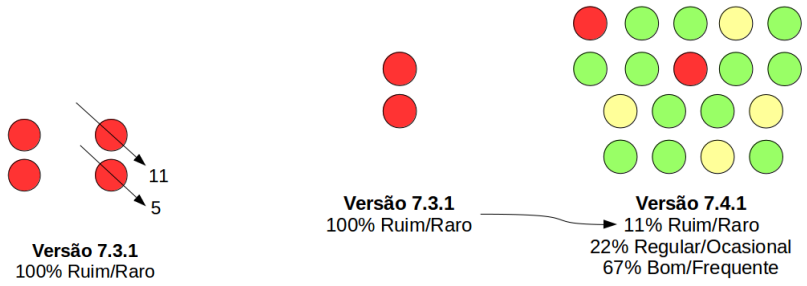
7.3.1 Número de Classes (NOC)

7.3.1.1 *JHotdraw*

- **Versões 7.3.1/7.4.1:** na Versão 7.3.1, no contexto da reestruturação, havia 2 pacotes classificados como *Ruim/Raro* pelo valor referência sugerido para a métrica Número de Classes (100% dos pacotes classificados como *Ruim/Raro*). Essa parte do sistema foi reestruturada em 18 pacotes na Versão 7.4.1, sendo: 2 *Ruim/Raro* (11%), 4 *Regular/Ocasional* (22%), 12 *Bom/Frequente* (67%). A Figura 7.2a ilustra a reestruturação. Os círculos representam os pacotes e as cores, as faixas em que eles foram classificados, vermelho: *Ruim/Raro*, amarelo: *Regular/Ocasional* e verde: *Bom/Frequente*. As setas partindo o círculo indicam que o pacote foi reagrupado em n subpacotes. A Figura 7.2b ilustra o contexto da reestruturação nas versões base e reestruturada, bem como a porcentagem de

[7] `http://www.eclipsecon.org/2004/EclipseCon_2004_TechnicalTrackPresentations/11_Edgar.pdf`

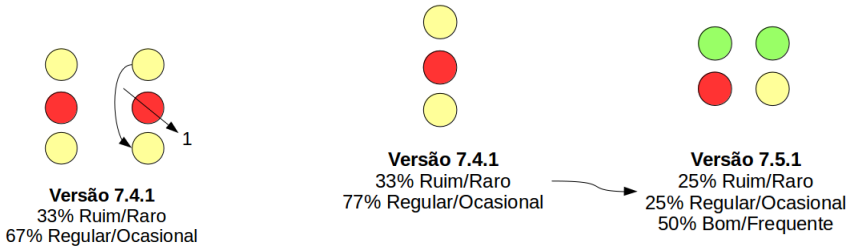
pacotes em cada uma das faixas. Nenhum pacote pré-existente saiu de uma faixa de classificação melhor para pior e a porcentagem de pacotes na faixa *Ruim/Raro* foi menor (100% para 11%) na versão reestruturada. Logo, os dados observados testemunham contra a hipótese nula.



(a) (b)

Figura 7.2: JHotdraw 7.3.1/7.4.1 (NOC): (a) Reestruturação (b) Classificação Pacotes

- **Versões 7.4.1/7.5.1:** na Versão 7.4.1, no contexto da reestruturação, havia 3 pacotes, sendo: 1 *Ruim/Raro* (33%) e 2 *Regular/Ocasional* (67%). Foi adicionado 1 pacote resultante do reagrupamento de classes em um desses pacotes e os outros dois tiveram classes movidas de um pacote para outro, conforme observado na Figura 7.3a. Essa parte do sistema foi reestruturada em 4 pacotes na Versão 7.5.1, sendo: 1 *Ruim/Raro* (25%), 1 *Regular/Ocasional* (25%) e 2 *Bom/Frequente* (50%). A Figura 7.3b ilustra o contexto da reestruturação nas versões base e reestruturada. Nenhum pacote pré-existente saiu de uma faixa de classificação melhor para pior e a porcentagem de pacotes na faixa *Ruim/Raro* foi menor (33% para 25%) na versão reestruturada. Logo, os dados observados testemunham contra a hipótese nula.



(a) (b)

Figura 7.3: JHotdraw 7.4.1/7.5.1 (NOC): (a) Reestruturação (b) Classificação Pacotes

- **Versões 7.5.1/7.6:** na Versão 7.5.1, no contexto da reestruturação, havia 1 pacote classificado como *Ruim/Raro* (100%). Foi adicionado 1 pacote resultante do reagrupamento de classes desse pacote, conforme observado na Figura 7.4a. Essa parte do sistema foi reestruturada em 2 pacotes na Versão 7.6, sendo 1 1 *Ruim/Raro* (50%) e 1 *Bom/Frequente* (50%). A Figura 7.4b ilustra o contexto da reestruturação nas versões base e reestruturada. Nenhum pacote pré-existente saiu de uma faixa de classificação melhor para pior e a porcentagem de pacotes na faixa *Ruim/Raro* foi menor (100% para 50%) na versão reestruturada. Logo, os dados observados testemunham contra a hipótese nula.

(a) (b)

Figura 7.4: JHotdraw 7.5.1/7.6 (NOC): (a) Reestruturação (b) Classificação Pacotes

7.3.1.2 *Eclipse*

Na Versão 2.1.3 (versão base) do *Eclipse* havia 442 pacotes, sendo: 202 *Bom/Frequente* (45,7%), 146 *Regular/Ocasional* (33,03%) e 94 *Ruim/Raro* (21,27%). Na Versão 3.0 (versão reestruturada), havia 668, sendo: 330 *Bom/Frequente* (49,4%), 217 *Regular/Ocasional* (32,49%) e 121 *Ruim/Raro* (18,11%). A porcentagem de pacotes na faixa *Ruim/Raro* foi menor, 21,27% para 18,11%. Isso é considerável, haja vista que houve um acréscimo de 282 pacotes na versão reestruturada. Se forem considerados somente esses 282 pacotes resultantes do processo de reestruturação, temos que apenas 24 (8,51%) foram classificados como *Ruim/Raro*, enquanto 171 (60,64%) foram classificados como *Bom/Frequente*. Logo, os dados observados testemunham contra a hipótese nula.

7.3.1.3 Conclusão

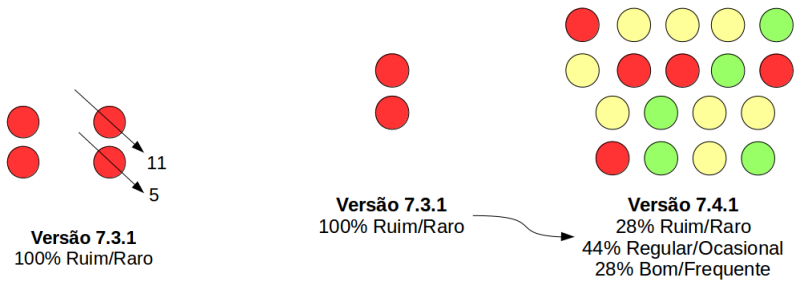
Em todos os casos estudados, o valor referência sugerido para a métrica Número de Classes foi capaz de medir o acréscimo da qualidade de software com a reestruturação dos pacotes. Logo, rejeita-se a hipótese nula, haja vista que a porcentagem de pacotes avaliados pela métrica na faixa *Ruim/Raro* foi menor na versão reestruturada do que

na versão base, no contexto da reestruturação, em todos os casos. Isso sugere a validade do valor referência proposto.

7.3.2 Acoplamento Aferente (CA)

7.3.2.1 *JHotdraw*

- **Versões 7.3.1/7.4.1:** na Versão 7.3.1, no contexto da reestruturação, havia 2 pacotes classificados como *Ruim/Raro* pelo valor referência sugerido para a métrica Acoplamento Aferente (100% dos pacotes classificados como *Ruim/Raro*). Essa parte do sistema foi reestruturada em 18 pacotes na Versão 7.4.1, sendo: 5 *Ruim/Raro* (28%), 8 *Regular/Ocasional* (44%), 5 *Bom/Frequente* (67%). A Figura 7.5a ilustra a reestruturação. A Figura 7.5b ilustra o contexto da reestruturação nas versões base e reestruturada, bem como a porcentagem de pacotes em cada uma das faixas. Nenhum pacote pré-existente saiu de uma faixa de classificação melhor para pior e a porcentagem de pacotes na faixa *Ruim/Raro* foi menor (100% para 28%) na versão reestruturada. Logo, os dados observados testemunham contra a hipótese nula.



(a) (b)

Figura 7.5: JHotdraw 7.3.1/7.4.1 (CA): (a) Reestruturação (b) Classificação Pacotes

- **Versões 7.4.1/7.5.1:** na Versão 7.4.1, no contexto da reestruturação, havia 3 pacotes, sendo: 1 *Ruim/Raro* (33%) e 2 *Regular/Ocasional* (67%). A Figura 7.6a ilustra a reestruturação, que resultou em 4 pacotes na Versão 7.5.1, sendo: 1 *Ruim/Raro* (25%), 1 *Regular/Ocasional* (25%) e 2 *Bom/Frequente* (50%). A Figura 7.6b ilustra o contexto da reestruturação nas versões base e reestruturada. Nenhum pacote pré-existente saiu de uma faixa de classificação melhor para pior e a porcentagem de pacotes na faixa *Ruim/Raro* foi menor (33% para 25%) na versão reestruturada, bem como a porcentagem de pacotes *Regular/Ocasional*

(67% para 50%). Logo, os dados observados testemunham contra a hipótese nula.

- **Versões 7.5.1/7.6:** na Versão 7.5.1, no contexto da reestruturação, havia 1 pacote classificado como *Ruim/Raro* (100%). A Figura 7.7a ilustra a reestruturação, que resultou em 2 pacotes na Versão 7.6, sendo 1 *Ruim/Raro* (50%) e 1 *Bom/Frequente* (50%). A Figura 7.7b ilustra o contexto da reestruturação nas versões base e reestruturada. Nenhum pacote pré-existente saiu de uma faixa de classificação melhor para pior e a porcentagem de pacotes na faixa *Ruim/Raro* foi menor (100% para 50%) na versão reestruturada. Logo, os dados observados testemunham contra a hipótese nula.

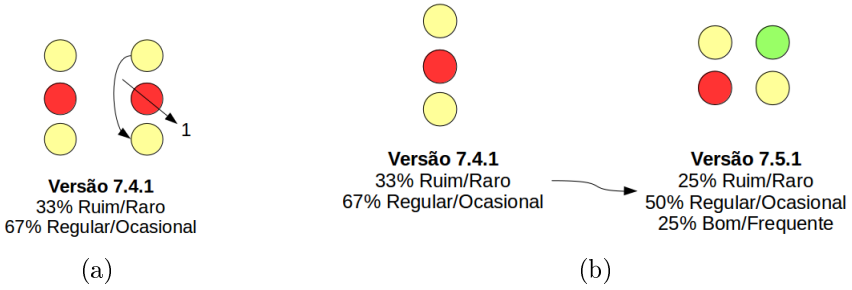


Figura 7.6: JHotdraw 7.4.1/7.5.1 (CA): (a) Reestruturação (b) Classificação Pacotes

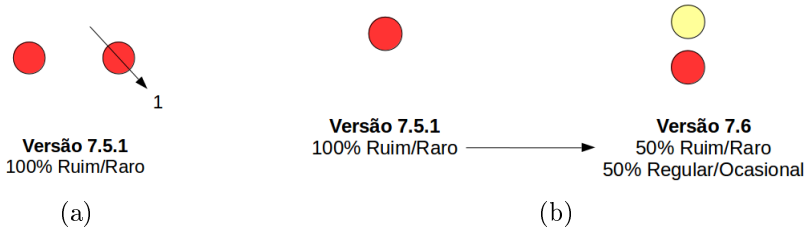


Figura 7.7: JHotdraw 7.5.1/7.6 (CA): (a) Reestruturação (b) Classificação Pacotes

7.3.2.2 *Eclipse*

Na Versão 2.1.3 (versão base) do *Eclipse* havia 442 pacotes, sendo: 191 *Bom/Frequente* (43,21%), 141 *Regular/Ocasional* (31,9%) e 110 *Ruim/Raro* (24,89%). Na Versão 3.0 (versão reestruturada), havia 668, sendo: 290 *Bom/Frequente* (43,41%), 223 *Regular/Ocasional* (33,38%) e 155 *Ruim/Raro* (23,2%). A porcentagem de pacotes na faixa *Ruim/Raro* foi menor (24,89% para 23,2%). Isso é considerável, haja vista que houve um acréscimo de 282 pacotes na versão reestruturada. Se forem considerados

somente esses 282 pacotes resultantes do processo de reestruturação, temos que apenas 26 (9,22%) foram classificados como *Ruim/Raro*, enquanto que 166 (58,87%) foram classificados como *Bom/Frequente*. Logo, os dados observados testemunham contra a hipótese nula.

7.3.2.3 Conclusão

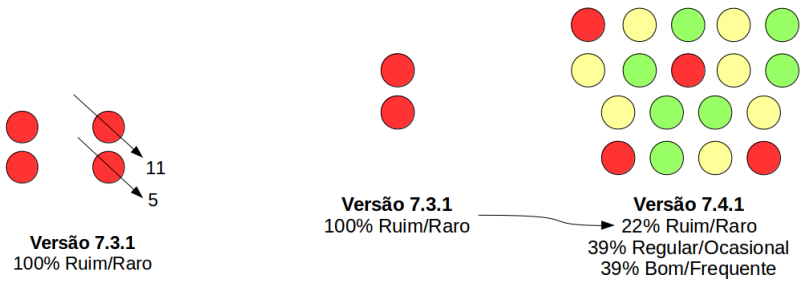
Em todos os casos estudados o valor referência sugerido para Acoplamento Aferente foi capaz de medir o acréscimo da qualidade de software com a reestruturação dos pacotes. Logo, rejeita-se a hipótese nula, haja visto que a porcentagem de pacotes avaliados pela métrica na faixa *Ruim/Raro* foi menor na versão reestruturada do que na versão base, no contexto da reestruturação, em todos os casos. Isso sugere a validade do valor referência definido.

7.3.3 Acoplamento Eferente (CE)

7.3.3.1 *JHotdraw*

- **Versões 7.3.1/7.4.1:** na Versão 7.3.1, no contexto da reestruturação, havia 2 pacotes classificados como *Ruim/Raro* pelo valor referência sugerido para a métrica Acoplamento Eferente (100% dos pacotes classificados como *Ruim/Raro*). Essa parte do sistema foi reestruturada em 18 pacotes na Versão 7.4.1, sendo: 4 *Ruim/Raro* (22%), 7 *Regular/Ocasional* (39%), 7 *Bom/Frequente* (39%). A Figura 7.8a ilustra a reestruturação. A Figura 7.8b ilustra o contexto da reestruturação nas versões base e reestruturada, bem como a porcentagem de pacotes em cada uma das faixas. Nenhum pacote pré-existente saiu de uma faixa de classificação melhor para pior e a porcentagem de pacotes na faixa *Ruim/Raro* foi menor (100% para 22%) na versão reestruturada. Logo, os dados observados testemunham contra a hipótese nula.

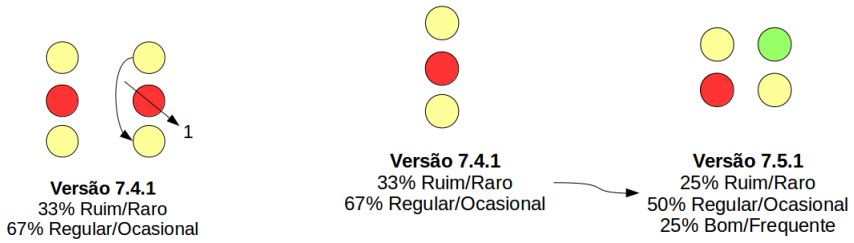
- **Versões 7.4.1/7.5.1:** na Versão 7.4.1, no contexto da reestruturação, havia 3 pacotes classificados, sendo: 1 *Ruim/Raro* (33%) e 2 *Regular/Ocasional* (67%). A Figura 7.9a ilustra a reestruturação, que resultou em 4 pacotes na Versão 7.5.1, sendo: 1 *Ruim/Raro* (25%), 2 *Regular/Ocasional* (50%) e 1 *Bom/Frequente* (25%). A Figura 7.9b ilustra o contexto da reestruturação nas versões base e reestruturada. Nenhum pacote pré-existente saiu de uma faixa de classificação melhor para pior e a porcentagem de pacotes na faixa *Ruim/Raro* foi menor (33% para 25%) na versão reestruturada, bem como a porcentagem de pacotes



(a) (b)

Figura 7.8: JHotdraw 7.3.1/7.4.1 (CE): (a) Reestruturação (b) Classificação Pacotes

Regular/Ocasional (67% para 50%). Logo, os dados observados testemunham contra a hipótese nula.



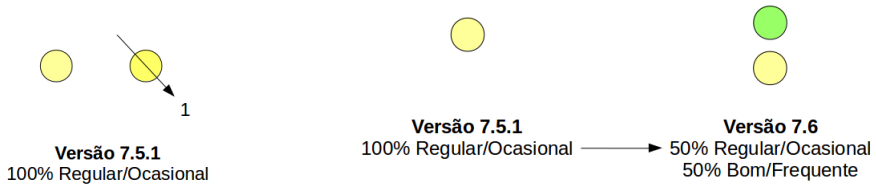
(a) (b)

Figura 7.9: JHotdraw 7.4.1/7.5.1 (CE): (a) Reestruturação (b) Classificação Pacotes

- **Versões 7.5.1/7.6:** na Versão 7.5.1, no contexto da reestruturação, havia 1 pacote classificado como *Regular/Ocasional* (100%). A Figura 7.10a ilustra a reestruturação, que resultou em 2 pacotes na Versão 7.6, sendo 1 *Regular/Ocasional* (50%) e 1 *Bom/Frequente* (50%). A Figura 7.10b ilustra o contexto da reestruturação nas versões base e reestruturada. Nenhum pacote pré-existente saiu de uma faixa de classificação melhor para pior e a porcentagem de pacotes na faixa *Regular/Ocasional* foi menor (100% para 50%) na versão reestruturada. Logo, os dados observados testemunham contra a hipótese nula.

7.3.3.2 *Eclipse*

Na Versão 2.1.3 (versão base) do *Eclipse* havia 442 pacotes, sendo: 181 *Bom/Frequente* (40,95%), 138 *Regular/Ocasional* (31,22%) e 123 *Ruim/Raro* (27,83%). Na Versão 3.0 (versão reestruturada), havia 668, sendo: 301 *Bom/Frequente* (45,06%), 206 *Regular/Ocasional* (30,84%) e 161 *Ruim/Raro* (24,1%). A porcentagem de pacotes na faixa *Ruim/Raro* foi menor (27,83% para 24,1%). Isso



(a) (b)

Figura 7.10: JHotdraw 7.5.1/7.6 (CE): (a) Reestruturação (b) Classificação Pacotes

é considerável, haja visto que houve um acréscimo de 282 pacotes na versão reestruturada. Se forem considerados somente esses 282 pacotes resultantes do processo de reestruturação, temos que apenas 35 (12,41%) foram classificados como *Ruim/Raro*, enquanto que 159 (56,38%) foram classificados como *Bom/-Frequente*. Logo, os dados observados testemunham contra a hipótese nula.

7.3.3.3 Conclusão

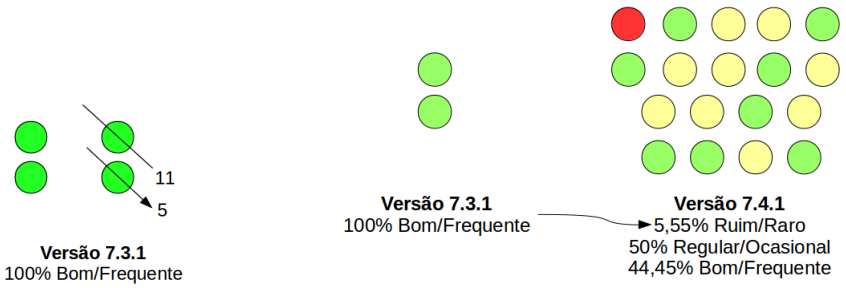
Em todos os casos estudados o valor referência sugerido para Acoplamento Eferente foi capaz de medir o acréscimo da qualidade de software com a reestruturação dos pacotes. Logo, rejeita-se a hipótese nula, haja visto que a porcentagem de pacotes avaliados pela métrica na faixa *Regular/Ocasional* foi menor na versão reestruturada do que na versão base, no contexto da reestruturação, em todos os casos. Isso sugere a validade do valor referência proposto.

7.3.4 Distância Normalizada (RMD)

7.3.4.1 *JHotdraw*

- **Versões 7.3.1/7.4.1:** na Versão 7.3.1, no contexto da reestruturação, havia 2 pacotes classificados como *Bom/Frequente* pelo valor referência sugerido para a métrica Distância Normalizada (100% dos pacotes classificados como *Bom/-Frequente*). Essa parte do sistema foi reestruturada em 18 pacotes na Versão 7.4.1, sendo: 1 *Ruim/Raro* (5,55%), 9 *Regular/Ocasional* (50%), 8 *Bom/Frequente* (44,45%). A Figura 7.11a ilustra a reestruturação. A Figura 7.11b ilustra o contexto da reestruturação nas versões base e reestruturada, bem como a porcentagem de pacotes em cada uma das faixas. A porcentagem de pacotes na faixa *Ruim/Raro* foi maior (0% para 5,55%) na versão reestruturada, além da porcentagem de pacotes na faixa *Bom/Frequente* ter sido menor (100% para 44,45%). Além disso, o valor referência não foi capaz de identificar a má qualidade nos

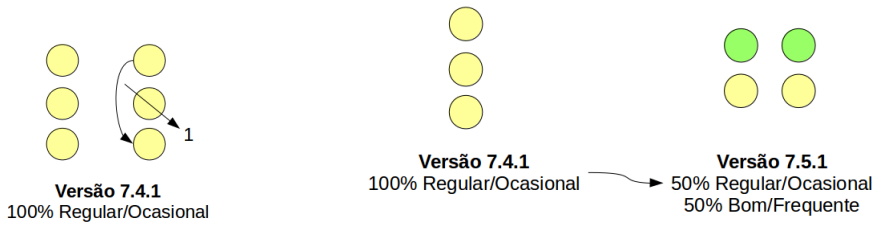
pacotes reestruturados da versão base, pois classificou os dois pacotes reestruturados como *Bom/Frequente*. Logo, os dados observados testemunham a favor da hipótese nula.



(a) (b)

Figura 7.11: JHotdraw 7.3.1/7.4.1 (RMD): (a) Reestruturação (b) Classificação Pacotes

- **Versões 7.4.1/7.5.1:** na Versão 7.4.1, no contexto da reestruturação, havia 3 pacotes classificados como *Regular/Ocasional* (100%). A Figura 7.12a ilustra a reestruturação, que resultou em 4 pacotes na Versão 7.5.1, sendo: 2 *Regular/Ocasional* (50%) e 2 *Bom/Frequente* (50%). A Figura 7.12b ilustra o contexto da reestruturação nas versões base e reestruturada. Nenhum pacote pré-existente saiu de uma faixa de classificação melhor para pior e a porcentagem de pacotes na faixa *Regular/Ocasional* foi menor (100% para 50%) na versão reestruturada. Logo, os dados observados testemunham contra a hipótese nula.



(a) (b)

Figura 7.12: JHotdraw 7.4.1/7.5.1 (RMD): (a) Reestruturação (b) Classificação Pacotes

- **Versões 7.5.1/7.6:** na Versão 7.5.1, no contexto da reestruturação, havia 1 pacote classificado como *Regular/Ocasional* (100%). A Figura 7.13a ilustra a reestruturação, que resultou em 2 pacotes na Versão 7.6, ambos *Regular/Ocasional* (100%). A Figura 7.13b ilustra o contexto da reestruturação nas versões base e

reestruturada. A porcentagem de pacotes na faixa *Regular/Ocasional* foi igual na versão reestruturada. Logo, o valor referência não foi capaz de medir o acréscimo de qualidade e os dados observados testemunham a favor da hipótese nula.

(a) (b)

Figura 7.13: JHotdraw 7.5.1/7.6 (RMD): (a) Reestruturação (b) Classificação Pacotes

7.3.4.2 *Eclipse*

Na Versão 2.1.3 (versão base) do *Eclipse* havia 442 pacotes, sendo: 297 *Bom/Frequente* (67,19%), 101 *Regular/Ocasional* (22,85%) e 44 *Ruim/Raro* (9,95%). Na Versão 3.0 (versão reestruturada), havia 668, sendo: 450 *Bom/Frequente* (67,37%), 155 *Regular/Ocasional* (23,2%) e 63 *Ruim/Raro* (9,43%). A porcentagem de pacotes na faixa *Ruim/Raro* foi menor (9,95% para 9,43%), porém, sem muita representatividade, haja visto que a porcentagem de pacotes em cada faixa variou pouco. Logo, não é possível afirmar que o valor referência foi capaz de medir o acréscimo de qualidade e que os dados observados testemunham a favor da hipótese nula.

7.3.4.3 Conclusão

Em apenas um dos casos estudados o valor referência sugerido para Distância Normalizada foi capaz de medir o acréscimo da qualidade de software com a reestruturação dos pacotes. No caso do *Eclipse*, o acréscimo medido, bem como a capacidade de medir a qualidade deteriorada na versão base, foi desconsiderável. Logo, não se pode rejeitar a hipótese nula. Tendo como premissa que a métrica Distância Normalizada é válida e apropriada para avaliar a qualidade de software do ponto de vista de pacotes, isso sugere que o valor referência proposto não é válido.

Tendo em vista que o processo de derivação utilizado para a identificação do valor referência para Distância Normalizada foi o mesmo das métricas Número de Classes, Acoplamento Aferente e Acoplamento Eferente, que tiveram resultados sugestivos em defesa dos valores referência, há de se levantar dúvidas acerca da relevância dessa métrica para avaliar pacotes de softwares orientados por objetos. Seus conceitos são, efetivamente, aplicados no desenvolvimento de software? Há preocupação real dos

engenheiros de software em relação aos conceitos de abstração e instabilidade que são trazidos pela métrica? A situação idealizada por Martin [1994] é, realmente, relevante para a avaliação quantitativa dos pacotes? Os conceitos estabelecidos como base para a definição da métrica não são disseminados como boas práticas para alcançar a qualidade de software tal como são os conceitos de acoplamento e coesão.

7.4 Conclusão

Com o objetivo de avaliar valores referência para métricas de pacote orientados por objetos, foi conduzido um experimento para verificar a capacidade de esses valores em medir o acréscimo de qualidade em processos de reestruturação. Isso sugere que o valor referência é capaz de indicar, quantitativamente, uma qualidade deteriorada em uma versão base de software em favor de uma qualidade aumentada em uma versão reestruturada. O experimento foi conduzido com 3 amostras de reestruturação local e uma amostra de reestruturação global em dois softwares amplamente conhecidos: *JHotDraw* e *Eclipse*. Os resultados dos experimentos sugerem a eficácia dos valores referência propostos para as métricas Número de Classes, Acoplamento Aferente e Acoplamento Aferente, bem como a ineficácia do valor referência sugerido para Distância Normalizada. A ineficácia avaliada para a métrica Distância Normalizada, dado o processo de derivação, que considerou uma amostra de 111 softwares orientados por objetos, levanta dúvidas sobre a qualidade da métrica como real indicativo quantitativo de qualidade estrutural do software.

7.5 Ameaças à Validade

A representatividade da amostra utilizada no experimento para softwares em geral é uma ameaça aos resultados obtidos nesse experimento. Além disso, trabalhar com reestruturações locais constitui uma ameaça à validade interna, pois, o ideal seria ter trabalhado com evoluções destinadas unicamente a reestruturação. Uma ameaça externa à validade do experimento é que os esforços de reestruturação podem não ter sucesso e resultar em um decréscimo da qualidade do software.

Capítulo 8

Avaliação dos Valores Referência em um Software Proprietário

Este capítulo descreve os resultados da avaliação dos valores referência identificados utilizando-se um software proprietário com qualidade interna deteriorada. Pelas características do objeto do estudo, espera-se que os valores referência propostos para as métricas estudadas sejam capazes de indicar a real situação do software. Seção 8.1 apresenta o objeto deste estudo e sua avaliação qualitativa. O estudo é dividido em três partes: na duas primeiras, descritas nas Seções 8.2 e 8.3, são avaliados os valores referência de métricas de métodos e de classes, respectivamente; na terceira parte, descrita na Seção 8.4, investigou-se se os valores referência propostos são úteis para identificar *bad smells* reportados por uma ferramenta conhecida para esse propósito. Seção 8.5 traz a conclusão.

8.1 Objeto do Estudo

Neste estudo, foi utilizado um software proprietário e de missão crítica de uma organização pública. Por razões de privacidade, o denominaremos ficticiamente de XYZ. A Tabela 8.1 sumariza algumas características quantitativas sobre esse software.

Tabela 8.1: Estatísticas acerca do software estudado.

Métrica	**Valor**
Total de Linhas de Código	54.297
Número de Classes	603
Número de Pacotes	139
Número de Métodos	2.532

Dentro da equipe que conduziu o desenvolvimento e que, agora, conduz a manutenção e evolução de XYZ, é consenso que, hoje, ele possui problemas na sua qualidade interna, que foi se deteriorando ao longo do tempo. Desde a sua implantação em 2009, não houve execução de manutenções preventivas, tampouco a condução de refatorações para melhorar a qualidade interna de XYZ. Atualmente, XYZ vem sofrendo manutenções constantes, sendo elas:

1. corretivas: visando corrigir erros de execução ou de requisitos de sistema;

2. evolutivas: devido à característica do negócio, manutenções evolutivas que trabalham na adição de novas funcionalidades ou alteração nas funcionalidades já existentes, decorrentes principalmente de novas legislações ou até mesmo alterações na legislação vigente, fazem do software XYZ objeto de constante mudança;

3. perfectivas: com o objetivo de melhorar a velocidade de processamento do software, são realizadas constantes alterações, dada a sua missão crítica;

4. adaptativas: por ser um sistema que está em operação há mais de cinco anos, ocorrem constantes evoluções no ambiente operacional que lhe exigem adequações.

Como XYZ está em constante mutação, é natural que suas estruturas internas se tornem mais complexas. De acordo com Lehman [1978], isso é parte natural da evolução de software. No entanto, à medida que os softwares crescem e sofrem manutenções, sua complexidade aumenta e a sua qualidade decresce. Nesse caso, recursos extras deveriam ser alocados na preservação e simplificação da estrutura do software, o que não é o caso de XYZ, uma vez que ele não recebeu esse tipo de reestruturação durante os mais de cinco anos de operação. Nesse sentido, acredita-se que suas estruturas internas apresentem qualidade deteriorada.

Com o objetivo de cruzar essas informações teóricas acerca dos aspectos que envolvem a manutenção e evolução de software e a situação real de XYZ, vista pelos profissionais que têm contato direto com o seu código-fonte, foram coletadas as opiniões de quatro membros da equipe de desenvolvedores acerca da qualidade interna de XYZ e os motivos pelos quais eles julgam que o software está nessa situação. Não foram utilizados formulários, nem direcionamentos específicos. Optou-se por deixar as opiniões fluirem naturalmente com o objetivo de caracterizar, pela visão qualitativa dos especialistas, a qualidade interna do software.

Pela análise dos relatos dos especialistas, percebeu-se um consenso de que o software estudado de fato possui qualidade interna deteriorada. Vários fatores foram ci-

tados como raiz desse problema que, no geral, envolvem a inexistência da adoção de metodologias que sistematizem os trabalhos de manutenção, a falta da figura do arquiteto de software, a análise de requisitos insuficiente e a falta de prioridade dada pela gerência à qualidade interna de software.

8.2 Parte 1 - Métodos

Não é objetivo deste estudo observar a qualidade global do software por meio de valores referência absolutos, mas, tendo em vista o consenso qualitativo de que o software apresenta problemas importantes em sua qualidade interna, espera-se que ele tenha uma maior porcentagem de métodos avaliados negativamente pelos valores referência em relação a maioria dos softwares.

A abordagem utilizada para a condução deste estudo foi estabelecer uma base com a qual seja possível comparar o sistema avaliado. Para tal, foram consideradas a porcentagem de métodos classificados na faixa *Bom/Frequente*, *Regular/Ocasional* e *Ruim/Raro* das métricas para os 111 sistemas presentes no Qualitas.*class* Corpus [Terra et al., 2013]. Posteriormente, os 112 sistemas, 111 do Qualitas.*class* Corpus e mais o sistema XYZ, foram ordenados da seguinte forma:

1. ordena-se de forma ascendente o conjunto de sistemas via a porcentagem de métodos classificados na faixa *Ruim/Raro*. Isso feito, obtém-se a posição do sistema avaliado no *ranking* obtido. Uma posição baixa no *ranking* significa que, relativamente, o sistema apresenta uma maior proporção de métodos classificados como *Ruim/Raro* e é pior em termos de qualidade quando é aplicada a métrica analisada na avaliação da qualidade do software por meio do valor referência sugerido.

2. Ordena-se de forma descendente o conjunto de sistemas por meio da porcentagem de métodos classificados na faixa *Bom/Frequente*. Uma posição baixa no *ranking* sugere que poucos métodos são bem avaliados pelos valores referência em relação aos outros softwares da amostra. Como a qualidade interna do software é considerada problemática, isso sugere poucos casos *falso-negativos*, ou seja, o software tem qualidade interna reconhecidamente degradada e seus métodos não foram bem avaliados pelos valores referência.

Para conduzir este estudo foram usadas as ferramentas acessórias construídas no processo de derivação para estabelecer as porcentagens dos sistemas do Qualitas.*class* Corpus. O software avaliado foi, então, medido por meio do *Metrics* e o arquivo

XML com as medições coletadas foi exportado. Utilizou-se a ferramenta *RAFTool* para contabilizar os artefatos classificados em cada uma das faixas para os valores referências sugeridos. A partir desses dados, formou-se o *ranking* para cada uma das métricas.

8.2.1 Resultados

Nesta seção são apresentados os resultados da Parte 1 do estudo realizado, mostrando e analisando os *rankings* identificados[1]. As métricas avaliadas são: Linhas de Código por Método, Profundidade de Blocos Aninhados, Número de Parâmetros e Complexidade de *McCabe*.

Linhas de Código por Método: pela primeira ordenação proposta, ordenação ascendente, o sistema avaliado ficou na 100ª posição, isto é, não foi pior que apenas 4 dos 111 sistemas restantes. Pela segunda ordenação proposta, ordenação descendente, o sistema avaliado ficou na 104ª posição, isto é, não foi pior que apenas 9 sistemas.

Profunfidade de Blocos Aninhados: pela primeira ordenação proposta, o sistema avaliado ficou na 92ª posição, ou seja, não foi pior que 20 dos 111 sistemas restantes. Pela segunda ordenação, o sistema avaliado ficou na 101ª posição, isso é, não foi pior que 11 sistemas.

Número de Parâmetros: pela primeira ordenação proposta, o sistema avaliado ficou na 107ª posição, isto é, não foi pior que apenas 5 dos 111 sistemas restantes. Pela segunda ordenação proposta, o sistema avaliado ficou na 105ª posição, isto é, não foi pior que apenas 7 sistemas.

Complexidade de *McCabe*: em ambas ordenações propostas, o sistema avaliado ficou na 103ª posição, isto é, não foi pior que apenas 9 sistemas.

8.2.2 Conclusão

Os dados mostram que o sistema XYZ possui, relativamente, menos métodos bem avaliados do que 90% dos sistemas presentes no Qualitas.*class* Corpus. Esse resultado indica que os valores referência propostos para as métricas de métodos foram capazes de refletir quantitativamente o cenário de qualidade degradada desse sistema. O sistema

[1] Os *rankings* para cada uma das métricas avaliadas estão disponíveis em: `http://homepages.dcc.ufmg.br/~tfilo/estudo_caso_3/`

analisado também possui, relativamente, um número elevado de métodos mal avaliados pelas métricas estudadas. Tudo isso contribui em direção ao entendimento de que os valores referência propostos para as métricas que avaliam os métodos não indicam qualidade onde há problemas, ou seja, não resultam em casos *falso-negativos*.

8.3 Parte 2 - Classes

Nesta parte do estudo caso, avaliou-se uma amostra de classes que foram apontadas como problemáticas do ponto de vista de qualidade interna pela equipe de desenvolvimento do software analisado. Os valores das métricas dessas classes foram coletados e analisados. Por meio dessa análise foi possível verificar a ocorrência de avaliações *falso-negativas*, uma vez que sabe-se previamente que as classes possuem problemas estruturais.

A Tabela 8.2 apresenta a amostra de classes utilizadas, com nomes fictícios, e a respectiva classificação obtida para os valores referência de NOF, DIT, WMC, NSC, NORM, LCOM, NOM e SIX, cujos resultados serão analisados da Seção 8.3.1 a Seção 8.3.8.

Tabela 8.2: Classificação obtida pela amostra de classes ruins do sistema XYZ.

-1 *Ruim/Raro*
0 *Regular/Ocasional*
+1 *Bom/Frequente*

Métrica / *Classe*	**NOF**	**DIT**	**WMC**	**NSC**	**NORM**	**LCOM**	**NOM**	**SIX**
LSI	+1	+1	-1	+1	+1	-1	-1	+1
NSI	+1	+1	-1	+1	+1	-1	-1	+1
RSI	+1	+1	-1	+1	+1	-1	-1	+1
NB	-1	0	-1	+1	-1	-1	-1	-1
NTB	-1	+1	-1	+1	+1	-1	-1	+1
RB	-1	+1	-1	+1	+1	-1	-1	+1
NDAO	+1	0	-1	+1	+1	+1	-1	+1
LDAO	+1	0	-1	+1	+1	+1	-1	+1

8.3.1 Número de Atributos (NOF)

Pela coluna NOF da Tabela 8.2, percebe-se que, da amostra, foram obtidas 7 classes bem avaliadas e 3 classes mal avaliadas. Como essas classes são definidas qualitativamente como problemáticas, elas foram inspecionadas com o objetivo de identificar o porquê dessa avaliação positiva, se é ou não correta, e a corretude da avaliação negativa.

- **LSI, NSI e RSI:** são classes do tipo *service*. De acordo com Fowler [2003], um modelo de domínio anêmico ocorre quando não é colocada lógica de negócio nos objetos de domínio. Em vez disso, existe uma série de objetos de serviço (*services*) que capturam essa lógica. Ao deslocar os comportamentos para os serviços, eles se tornam *Transaction Scripts*, que organizam a lógica de negócio por meio de procedimentos que tratam uma requisição da camada de visão. Esses *services* ficam no topo do modelo de domínio e utilizam o modelo como repositório de dados. Com isso, os objetos de domínio se tornam *"bags of getters and setters"*, não encapsulando lógica alguma referente aos dados. Os *services* são um agrupamento de funções procedurais relacionadas aos dados do domínio. Logo, os objetos anêmicos não possuem comportamentos e os *services* são agrupamentos de funções procedurais que utilizam os objetos anêmicos. Por tal motivo, essas classes do tipo *service* foram classificadas como *Bom/Frequente* pela métrica Número de Atributos, não por possuírem uma quantidade razoável de atributos, e sim por não possuírem atributos. Portanto, utilizando somente a métrica NOF, essas classes seriam bem avaliadas, apesar de serem problemáticas dentro do projeto, constituindo potenciais casos *falso-negativos*.

- **NDAO e LDAO:** são objetos do tipo DAO (*Data Acess Object*), que encapsulam toda a lógica de acesso a dados dentro de uma aplicação MVC (*Model-View-Controller*) [Oracle, 2014a]. São classes que, por seu objetivo e missão dentro da arquitetura, não possuem ou possuem poucos atributos, sem caracterizar uma violação arquitetural ou uma má prática de programação, tal como é a modelagem de domínio anêmica. Essas classes são problemáticas pelo seu tamanho e complexidade, e não pelo Número de Atributos. Logo, as classes foram bem avaliadas de forma correta em relação a Número de Atributos pelo valor referência proposto.

- **NB, NTB e RB:** são classes do tipo *managed bean*, que foram mal avaliadas pelo valor referência proposto para Número de Atributos. Esse tipo de classe é típica em aplicações que utilizam *JavaServer Faces*, sendo que cada uma delas deveria estar associada a componentes de uma determinada página *web* da aplicação [Oracle, 2014b]. Essas classes são grandes e complexas pois não estão sendo componentizadas em *managed beans* mais coesos e que atendam a um propósito específico dentro da página *web*. Percebe-se que cada *managed bean* armazena um atributo para a grande maioria dos dados de um formulário, que, por questões de regras de negócio, podem ter uma grande quantidade de entrada de dados. Logo, essas classes são mal projetadas, podendo ser mais bem subdividas em compo-

nentes mais coesos, que atendam a um propósito específico dentro do formulário. Um exemplo simples de melhoria refere-se ao preenchimento de um formulário no qual um desses campos é a seleção de um município pesquisado em uma janela. Essa seleção de município deveria ser um componente (*managed bean*) separado, e não "costurado" ao componente principal da tela em questão. Isso proporcionaria, por exemplo, a reutilização desse componente, caso outra tela tivesse o mesmo requisito, evitando código duplicado. Portanto, a avaliação qualitativa da classe condiz com o resultado da aplicação do valor referência, ambas sugerindo que a classe apresenta problemas do ponto de vista de sua estrutura e deve ser objeto de análise em relação ao excesso de atributos.

8.3.2 Profundidade de Árvore de Herança (DIT)

Pela coluna DIT da Tabela 8.2, percebe-se que três classes foram avaliadas como *Regular/Ocasional* e as demais foram bem avaliadas. No software, a maioria das classes não utiliza herança e, por tal motivo, são bem avaliadas pelo valor referência. De fato, não há problemas nessas classes relacionados a profundidade de árvore de herança e não é possível considerá-los casos *falso-negativos*, pelo contrário, o valor referência da métrica avaliou bem essas classes naquilo que a métrica se propõe e de forma correta.

Em relação às classes que foram avaliadas de forma regular, NDAO e LDAO possuem uma hierarquia de herança imposta pelo *framework* de persistência utilizado, com mais uma classe base que executa operações padrão de inserção, atualização, exclusão e leitura, acrescido da classe *Object* padrão da plataforma Java. Em uma avaliação qualitativa, entende-se que o valor referência avalia essa árvore como *Regular/Ocasional* de forma correta, pois qualquer adição de um novo nível de herança acima dessa classe, que tem DIT igual a 4, tornaria o seu entendimento difícil. Em relação à classe NB, que também possui DIT igual a 4, a situação é semelhante.

8.3.3 Métodos Ponderados por Classe (WMC)

Pela coluna WMC da Tabela 8.2, observa-se que todas as classes consideradas ruins da amostra foram classificadas como *Ruim/Raro*. WMC é uma medida de complexidade da classe, e, levando em conta as características do sistema XYZ e seu processo de evolução, isso comprova que o aumento da complexidade é um fator natural da evolução de software, conforme aponta Lehman [1978]. O valor referência para WMC sugerido mostrou-se capaz de avaliar esse cenário, de forma quantitativa. De fato, essas classes

do sistema são demasiadamente complexas e assumiram muitas responsabilidades ao logo da evolução do software, sendo classes de difícil entendimento e manutenção.

8.3.4 Número de Filhas (NSC)

A coluna NSC da Tabela 8.2 mostra que todas as classes foram bem avaliadas pelos valores referência de NSC. De fato, são classes que não possuem filhas, portanto, não possuem problemas relacionados aos aspectos avaliados pela métrica.

8.3.5 Número de Métodos Sobrescritos (NORM)

A coluna NORM da Tabela 8.2 mostra que todas as classes foram bem avaliadas pelos valores referência de NORM, com exceção da classe NB. De fato, pela baixa utilização do recurso da herança, mostrada pelos resultados de DIT e a análise do software, esse resultado era esperado, pois a classe não tem problemas relacionados a sobrescrita excessiva de métodos e, portanto, são bem avaliadas de forma correta. Para a classe NB, que foi avaliada como *Ruim/Raro*, é possível observar que essa classe foi uma das três que foram classificadas como regular por DIT, indicando uma pronfundidade de árvore de herança fora do ideal e próxima ao considerado impróprio. NB sobrescreve 7 métodos, uma quantidade considerada alta pelo valor referência da métrica. O grande problema dessa quantidade de métodos sobrescritos é que torna a classe de difícil entendimento, pois o comportamento das classes mãe, e, consequentemente, do sistema, podem utilizar esses métodos. Esse problema se agrava quando combinado a uma hierarquia de herança imprópria, podendo a classe estar sobrescrevendo muitos métodos, pois as classes mãe não são apropriadas ao seu propósito. Portanto, a avaliação negativa é concordante com a inspeção da classe, bem como as avaliações positivas, não caracterizando casos *falso positivos* ou *negativos*.

8.3.6 Ausência de Coesão em Métodos (LCOM)

A coluna LCOM na Tabela 8.2 mostra a classificação obtida pelas classes. É possível observar que 6 das classes são de baixa coesão. As classes do tipo *service* não mostram coesão entre os métodos pois não tem atributos, afinal, utilizam o modelo de domínio anêmico como repositório de dados. Logo, os *services* inspecionados tem uma avaliação negativa condizente em relação a baixa coesão que apresentam. Os objetos *managed beans*, por estarem mal componentizados, conforme descrito na Seção 8.3.1, apresentam baixa coesão. Caso fossem mais bem componentizados, o aumento da coesão seria natural, pois os métodos apresentariam maior similaridade. Por outro lado, os objetos

do tipo DAO apresentaram similaridade pelo fato de que a grande maioria dos métodos utilizam atributos herdados da classe mãe relacionada ao *framework* para efetuar operações no banco de dados. Logo, a avaliação positiva foi correta para esses casos.

8.3.7 Número de Métodos (NOM)

A coluna NOM mostra que todas as classes foram avaliadas como *Ruim/Raro* pelos valores referência de Número de Métodos, o que está de acordo com as avaliações realizadas pelo valor referência de WMC. Segundo Lehman [1980], as funcionalidades oferecidas por um sistema devem ser continuamente incrementadas de forma a manter a satisfação do usuário. Se isso não for bem gerenciado, projetado e planejado, a qualidade de software deteriora pois as classes se tornam grandes e complexas para agregar as novas funcionalidades, como no caso dessas classes analisadas. Como exemplo, NB possui 80 métodos. Aspectos de qualidade interna relacionados a modularidade, legibilidade e manutenabilidade são deteriorados ao extremo em uma classe com essa quantidade de métodos.

8.3.8 Índice de Especialização (SIX)

A coluna SIX mostra que 7 classes foram bem avaliadas e a classe restante foi avaliada como *Ruim/Raro*. Essa métrica visa avaliar o quanto determinada classe sobrescreve o comportamento de suas superclasses. Como era esperado, devido à relação de SIX com NORM e DIT (Seção 2.4), NB foi mal avaliada por SIX, tal como foi mal avaliada por NORM e avaliada regularmente por DIT. As outras classes não excedem os níveis normais de especialização sugeridos pelo valor referência de SIX e, de fato, elas não utilizam demasiadamente a sobrescrição do comportamento de suas superclasses.

8.3.9 Conclusão

Para esta análise, partiu-se de um conjunto de oito classes qualitativamente definidas como problemáticas do ponto de vista de seus fatores internos de qualidade, com o objetivo de realizar uma análise de contraponto em relação às avaliações obtidas por meio da aplicação dos valores referência propostos. O risco de avaliações *falso-negativas* torna-se temerário quando são aplicados os valores referência em um software de qualidade degradada. Uma avaliação *falso positiva* pode ser descartada por meio de uma inspeção manual nas classes mal avaliadas, porém, se uma classe ruim for bem avaliada pelo valor referência, ela poderia nem mesmo ser inspecionada quando da efetiva utilização de valores referência em um processo de medição de software.

Todas as classes obtiveram, pelo menos, 3 classificações fora da faixa *Bom/Frequente*. Isso sugere que, se o conjunto de métricas para as quais foram propostos valores referência fosse efetivamente aplicado no gerenciamento da qualidade, todas as classes da amostra deveriam ser objetos de inspeção por apresentarem valores para as métricas fora do padrão, ou seja, não haveria caso *falso-negativo*.

Portanto, conclui-se que idealmente uma única métrica não deve ser usada para definir a qualidade de uma classe, recomendando-se ser empregadas mais métricas para avaliá-la efetivamente. Dessa forma, casos *falso-negativos* podem ser minimizados quando os valores referência propostos nesse trabalho forem aplicados na avaliação da qualidade interna de softwares orientados por objetos. Essa conclusão está de acordo com outros achados descritos na literatura (Rosenberg et al. [1999], Ferreira et al. [2012]), que sugerem que uma única métrica não deve ser utilizada para avaliar a qualidade do código de forma geral, ou seja, quando não se deseja avaliar um atributo específico de qualidade.

8.4 Parte 3 - *Bad Smells*

Esta seção descreve a parte do estudo que tem por objetivo avaliar os valores referência por meio da verificação do uso deles para identificar *bad smells* [Fowler, 1999] em software. *JDeodorant*[2] é um *plugin* para o *Eclipse* que identifica *bad smells*, sugerindo refatorações para minimizá-los. A ferramenta foi utilizada para identificar, no sistema XYZ, a presença dos *bad smells God Class* e *Long Method*. Os métodos e classes de XYZ foram classificados por meio da aplicação dos valores referência. Com esses dados disponíveis foram cruzadas as duas informações de duas variáveis qualitativas: a presença ou ausência do *bad smell* e a classificação ou não classificação do método ou classe como *Bom/Frequente*. Obteve-se, então, via tabela, a quantidade de classes que possuem ou não o *bad smell* contra a quantidade de classes classificadas e não classificadas como *Bom/Frequente* pelo valor referência. Esse tipo de tabela é chamada de *Tabela de Contingência de Duas Variáveis Qualitativas* [R, 2014a]. Ela permite, por exemplo, identificar a quantidade de casos *falso-negativos*, isto é, quando o valor classifica o método ou a classe como *Bom/Frequente*, mas, na verdade, ela possui um *bad smell* associado àquela métrica.

As seguintes hipóteses foram levantadas em relação a presença do *bad smell* e a classificação obtida pelo método ou classe por meio do valor referência da métrica m:

[2] http://www.jdeodorant.com/

- H_m^{null}: a presença do *bad smell* independe do método ou da classe não ser classificado pela métrica m na faixa *Bom/Frequente*.

- $H_m^{alternativa}$: a presença do *bad smell* depende do método ou da classe não ser classificada pela métrica m na faixa *Bom/Frequente*.

Para avaliar as hipóteses, foi aplicado um teste estatístico que avalia a dependência entre variáveis qualitativas, chamado *Teste de Independência de Chi-squared* [R, 2014a], cuja entrada é a tabela de contingência de duas variáveis qualitativas. Como retorno, foi obtido o *p-value*, que representa a probabilidade de se obter uma estatística de teste igual ou mais extrema que aquela observada na amostra, sob a hipótese nula. Se o *p-value* for maior que 5%[3], não se rejeita a hipótese nula. Caso contrário, pode-se rejeitá-la, o que sugere que há uma relação entre a presença do *bad smell* e o método ou a classe não ser classificada como *Bom/Frequente*. Nesse processo, utilizou-se a ferramenta R para realizar o teste[4].

8.4.1 Resultados Obtidos para o *God Class*

De acordo com Lanza et al. [2007], a falha de projeto denominada *God Class* refere-se a classes que 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 e complexas e mostram baixa coesão. Por esse motivo, esse *bad smell* foi correlacionado com quatro métricas para as quais propomos valores referência: WMC, NOM e NOF, que avaliam a questão do tamanho e complexidade da classe, e para a métrica LCOM, que avalia a coesão. A seguir são apresentados os resultados para essas métricas.

As Tabelas de Contingência 8.3, 8.4, 8.5 e 8.6 sumarizam a quantidade de classes que possuem ou não o *bad smell God Class* contra a classificação *Bom/Frequente* dos valores referência das métricas WMC, NOM, LCOM e NOF, respectivamente. Aplicando o teste de independência de *Chi-squared* da ferramenta R em cada uma dessas tabelas obteve-se $p\text{-}value < 0,01 < 0,05$, o que indica a validade da associação com $\alpha = 1\%$ de nível de significância. Logo, rejeita-se a hipótese nula para cada uma dessas métricas.

[3]Menor nível de significância com que não se rejeita a hipótese nula, comumente adotado a 5%.

[4]Conforme o exemplo disponível em: `http://www.r-tutor.com/elementary-statistics/goodness-fit/chi-squared-test-independence`

Esses resultados sugerem que há uma baixa probabilidade de haver casos *falso-negativos* em relação a *God Classes* para os valores referência das métricas WMC, NOM, LCOM e NOF. Ou seja, é pouco provavel que quando algum dos valores dessas métricas em uma determinada classe correspondam à faixa *Bom/Frequente* de seus valores referência, a classe apresente o *bad smell God Class*.

Tabela 8.3: Tabela de contingência para WMC

Bom/Frequente? / ***God class?***	**Não**	**Sim**
Sim	53	6
Não	109	435

Tabela 8.4: Tabela de contingência para NOM

Bom/Frequente? / ***God class?***	**Não**	**Sim**
Sim	48	11
Não	46	498

Tabela 8.5: Tabela de contingência para LCOM

Bom/Frequente? / ***God class?***	**Não**	**Sim**
Sim	47	12
Não	71	473

Tabela 8.6: Tabela de contingência para NOF

Bom/Frequente? / ***God class?***	**Não**	**Sim**
Sim	38	21
Não	37	507

8.4.2 Resultados Obtidos para o *Long Method*

Segundo Lanza et al. [2007], o *bad smell Long Method* refere-se a métodos nos quais, no processo natural da evolução de software, são adicionadas mais e mais funcionalidades, tornando-o de difícil compreensão e manutenção. Esses métodos são excessivamente grandes (mostrado por MLOC), possuem um nível de aninhamento profundo (mostrado

por NBD) e possuem muitas caminhos de execução (mostrado por VG). Pasor esse motivo, esse *bad smell* foi correlacionado com as métricas: MLOC, NBD e VG. Os resultados são mostrados a seguir.

As Tabelas de Contingência 8.7, 8.8 e 8.9 sumarizam a quantidade de métodos que possuem ou não o *bad smell Long Method* contra a classificação *Bom/Frequente* dos valores referência das métricas MLOC, NBD e VG, respectivamente. Aplicando o teste de independência de *Chi-squared* da ferramenta R em cada uma dessas tabelas obteve-se *p-value* $< 0,01 < 0,05$, o que indica a validade da associação com $\alpha = 1\%$ de nível de significância. Logo, rejeita-se a hipótese nula para essas métricas.

Esses resultados sugerem que há uma baixa probabilidade de haver casos *falso-negativos* em relação ao *bad smell Long Method* para os valores referência das métricas MLOC, NBD e VG. Ou seja, é pouco provável que quando algum dos valores referência dessas métricas classifique um determinado método como *Bom/Frequente*, ele apresente o *bad smell Long Method*.

Tabela 8.7: Tabela de contingência para MLOC

Bom/Frequente? / ***Long Method?***	**Não**	**Sim**
Sim	296	83
Não	395	1749

Tabela 8.8: Tabela de contingência para NBD

Bom/Frequente? / ***Long Method?***	**Não**	**Sim**
Sim	324	55
Não	612	1532

Tabela 8.9: Tabela de contingência para VG

Bom/Frequente? / ***Long Method?***	**Não**	**Sim**
Sim	290	89
Não	433	1711

8.4.3 Conclusão

WMC, NOM, LCOM e NOF são as métricas para as quais foram propostos valores referência e que são relacionadas ao *bad smell God Class*. No entanto, para propor

Tabela 8.10: Tabela de contingência para DIT

God class? \ *Bom/Frequente?*	**Não**	**Sim**
Sim	7	52
Não	100	444

uma contra-prova para o que foi obtido em termos de significância estatística para as métricas relacionadas a esse *bad smell*, realizou-se o mesmo teste para a métrica DIT, que não tem relação direta com o *bad smell*. Tabela 8.10 mostra a tabela de contingência para essa métrica. Ao aplicar a função *chisq.test*, foi retornado $p\text{-}value = 0,2867 > 0,05$, o que não permite a rejeição da hipótese nula. Logo, não há dependência entre a ocorrência de *God Classes* e a classe não ser classificada na faixa *Bom/Frequente* pelo valor referência de DIT, conforme esperado.

Pelo exposto, foi possível verificar que há uma dependência estatística significante entre: (1) a classe ser bem avaliada pelos valores referência de WMC, NOM, LCOM e NOF e não possuir *bad smell God Class* e (2) o método ser bem avaliado pelos valores referência de MLOC, NBD e VG e não possuir o *bad smell Long Method*. Essas duas situações testemunham a favor dos valores referência sugeridos, pois indicam que casos *falso-negativos* são raros em relação a classes e métodos bem avaliados, o que é fundamental para que a classificação sugerida pelos valores referência seja confiável.

8.5 Conclusão

O objetivo deste estudo foi verificar se a aplicação dos valores referência identificam métodos e classes de qualidade quando realmente são, evitando assim um dos possíveis erros quando da utilização dos valores referência, que são avaliações *falso-negativas*. Avaliações *falso-negativas* refletem um panorama de qualidade que mascara problemas existentes. Nessa análise foi usado um software proprietário qualitativamente definido como de qualidade interna deteriorada e, por essa razão, esperava-se que ele não poderia ser bem avaliado pelos valores referência. Para verificar se os valores referência refletiriam esse panorama, a estratégia usada foi dividida em três partes.

A primeira parte do estudo mostrou que o sistema avaliado possui menos métodos bem avaliados, e consequentemente mais métodos mal avaliados, do que 90% dos sistemas presentes no Qualitas.*class* Corpus. A segunda parte do estudo partiu de uma amostra de 8 classes desse sistema, que foram apontadas previamente pelos especialistas como detentoras de problemas estruturais, e as respectivas classificações obtidas

pelos valores referência sugeridos para as métricas de classe. Os resultados revelaram que, das 8 métricas, pelo menos 3 não classificaram as classes como *Bom/Frequente*, o que indica que as classes não seriam bem avaliadas, evitando casos *falso-negativos*, e que uma única métrica não deve ser utilizada para definir o panorama de qualidade de uma classe, pois isso pode incorrer em casos *falso-negativos*. A terceira parte do estudo comprovou que há uma relação estatística significante entre a ocorrência de dois *bad smells* bem conhecidos, *God Class* e *Long Method*, e a não classificação de classes e métodos como *Bom/Frequente* pelos valores referência de métricas relacionadas a esses *bad smells*. Isso indica que a aplicação dos valores referência sugeridos permite a identificação dos *bad smells God Class* e *Long Method* com baixa probabilidade de ocorrência de casos *falso-positivos*.

Portanto, os resultados deste estudo sugerem que a aplicação dos valores referência propostos são confiáveis em termos da indicação quantitativa de boa qualidade dos métodos e classes avaliadas. Isso é fundamental para que a aplicação dos valores referência seja confiável não somente em suas avaliações negativas, mas também em suas avaliações positivas. Como consequência indireta do estudo, também foi possível mostrar que os valores referência são capazes de identificar métodos e classes com problemas estruturais, o que também reflete o panorama de qualidade deteriorada do software.

Capítulo 9

Avaliação dos Valores Referência em Softwares do Qualitas.*class* Corpus

Este capítulo apresenta a condução de dois estudos de casos. Estudo de Caso 2 está associado a métodos e classes de baixa qualidade. Estudo de Caso 3 avalia um software de alta qualidade. O objetivo de ambos é verificar a capacidade de os valores referência constatar esses panoramas. Os dois estudos de casos foram conduzidos nos softwares do Qualitas.*class* Corpus [Terra et al., 2013], *dataset* usado no processo de derivação dos valores referência.

9.1 Estudo de Caso 2

O objetivo deste estudo é avaliar a capacidade dos valores referência em indicar métodos e classes com a qualidade deteriorada. Para a realização deste experimento foram considerados como alvo todos os métodos e classes do *dataset* de sistemas usado no processo de derivação. Como o alvo são todos esses sistemas, não foi possível a utilização da *RAFTool*, pois ela trabalha com um único sistema por vez. Logo, foi usado o próprio banco de dados criado no processo de derivação, que armazena as medições usadas. O estudo consistiu em inspecionar os métodos e classes classificados na faixa *Ruim/Regular* com o propósito de comparar a avaliação qualitativa com a avaliação quantitativa.

9.1.1 Resultado da Inspeção de Métodos

O Qualitas.*class* Corpus contém de 1.663.526 métodos, o que torna difícil a seleção de uma amostra viável para conduzir a inspeção, mesmo quando utiliza-se uma estratégia de composição de métricas. Dessa forma, aplicamos o filtro mais restritivo possível, que selecionou todos os métodos avaliados na faixa *Ruim/Raro* pelas métricas de métodos presentes no catálogo: Linhas de Código por Método, o que resultou em 80.061 métodos; dos quais, Complexidade de *McCabe*, o que resultou em 63.030; Número de Parâmetros, o que resultou em 5.531; e Profundidade de Blocos Aninhados, o que resultou em 3.858.

Mesmo com a utilização de todos os valores referência definidos, não foi possível alcançar um número razoável de métodos para inspeção manual, afinal, 3.858 continua sendo um número muito alto, apesar de essa quantidade representar, de forma relativa, apenas 0,23% do total de métodos da amostra. Portanto, ainda fez-se necessário selecionar, dentre esses 3.858 métodos, uma amostra viável a ser inspecionada manualmente. Para tornar o estudo mais reprodutível, descrevemos, a seguir, o método adotado para selecionar, aleatoriamente, 10 métodos para serem avaliados dentre os 3.858 identificados por meio da aplicação dos valores referência:

1. Para se chegar aos 3.858 métodos, fez-se a interseção dos métodos classificados como *Ruim/Raro* por MLOC, VG, NBD e PAR. Esse resultado pode ser reproduzido por meio da execução do seguinte comando SQL no banco de dados criado para o processo de derivação e descrito na Seção 4.1.2.1.

```
1  SELECT
2   name, source, package, source_id
3  FROM
4  ( (SELECT name, source, package, source_id FROM
       measure where metric_id = 'MLOC' AND value > 30)
5   UNION ALL
6   (SELECT name,source,package, source_id FROM measure
       where metric_id = 'VG' AND value > 4)
7   UNION ALL
8   (SELECT name, source, package, source_id FROM measure
       where metric_id = 'PAR' AND value > 4)
9   UNION ALL
10  (SELECT name, source, package, source_id FROM measure
       where metric_id = 'NBD' AND value > 3)
```

```
11 ) as identified_uncommon_methods
12 GROUP BY
13   name, source, package, source_id
14 HAVING
15   count(*) = 4
16 ORDER BY
17   name;
```

2. O retorno da consulta foi ordenado pelo nome do método (coluna *name*). Nota-se que a ordenação não tem relação com o valor absoluto de nenhuma métrica usada na composição.

3. Foi utilizado um algoritmo de geração de números aleatórios para gerar 10 números entre 0 e 3.858. Foram gerados os seguintes números: 92, 205, 963, 1122, 1376, 2683, 2882, 3063, 3417, 3727.

4. Por meio da ordenação e dos números aleatórios gerados, que representam posições na lista ordenada de métodos obtida, obtém-se o n-ésimo método da lista.

5. Com o conhecimento dos números aleatórios e o critério de ordenação estabelecido, pode-se chegar aos mesmos 10 métodos aqui selecionados.

Por meio desse critério, os métodos selecionados podem ser de qualquer um dos 111 sistemas presentes no Qualitas.*class* Corpus. Uma vez identificados, é necessário que o projeto compilável em que o método está presente seja importado para o *Eclipse* de forma a possibilitar a sua análise.

Tabela 9.1 mostra os métodos selecionados, exibindo seu nome, classe e sistema. A seguir é descrita a avaliação qualitativa de cada um desses métodos.

- **addGetterSetterChanges:** esse método possui 34 linhas de código, Complexidade de *McCabe* igual a 9, possui 5 parâmetros e possui Profundidade de Blocos Aninhados igual a 4. Dois de seus parâmetros são duas variáveis do tipo *boolean*, *usingLocalGetter* e *usingLocalSetter*. Esses mesmos parâmetros são utilizados em outros dois métodos da classe: *checkInHierarchy* e *checkMethodNames*. Percebe-se que eles são relacionados, podendo ser substituídos por *Parameters Objects*, tal como descrito na Seção 5.2.11. Se isso fosse feito, o método seria classificado como *Regular/Ocasional*, pois passaria a ter 4 parâmetros. O nome do método refere-se a adição de mudanças, no plural, o que sugere que o método tem como responsabilidade realizar mais de uma mudança relacionada a adição de *getters* e *setters*

Tabela 9.1: Métodos inspecionados manualmente.

Método	Classe	Sistema
addGetterSetterChanges	SelfEncapsulateFieldRefactoring	eclipse-3.7.1
analyseAssignment	QualifiedNameReference	eclipse-3.7.1
diffList	CasualDiff	netbeans-7.3
drawSubCategoryLabels	SubCategoryAxis	jfreechart-1.0.13
findLocalMethods	CompletionEngine	aspectj-1.6.9
prepMinion	GenericStatement	derby-10.9.1.0
readAj5ClassAttributes	AtAjAttributes	aspectj-1.6.9
REPTree.Tree#buildTree	REPTree	weka-3-6-9
startFileInternal	FSNamesystem	hadoop-1.1.2
visitBranchInstruction	CodeSubroutineInliner	proguard-4.9

para o refatoramento do tipo *SelfEncapsulatedField*, que é responsabilidade da classe. Além disso, o método possui, claramente, dois trechos de código-fonte que realizam tarefas distintas, que poderiam ser exportados para dois métodos menores, o que o enquadraria fora da faixa *Ruim/Raro* para as quatro métricas utilizadas.

- **analyseAssignment:** esse método possui 89 linhas de código, Complexidade de *McCabe* igual a 36, possui 5 parâmetros e possui Profundidade de Blocos Aninhados igual a 4. Esse método possui, de início, um *switch-case* com dois *cases* nos quais o código executado possui mais de 10 linhas e está profundamente aninhado. Esses trechos poderiam ser exportados para dois novos métodos mais singulares. O método possui uma série de *ifs* em sequência, cujos trechos de código poderiam ser exportados para outros métodos definindo uma única responsabilidade. Além disso, o método possui muitos *ifs* dentro de *loops* que não contribuem com o entendimento do código. Tudo isso mostra que, devido ao fato do método efetuar inúmeras tarefas, ele é bem complexo, o que o induz a ter uma avaliação qualitativa ruim.

- **diffList:** esse método possui 163 linhas de código, Complexidade de *McCabe* igual a 54, possui seis parâmetros e possui Profundidade de Blocos Aninhados igual a 8. Esse método possui um *switch-case* com quatro *cases* nos quais o código executado possui muitas linhas, e estão profundamente aninhados. Esses trechos poderiam ser exportados para quatro novos métodos mais bem definidos e mais simples. Outro ponto importante em termos de qualidade desse método é que ele possui cinco pontos de retorno, justamente pelo fato do método ter mais de uma responsabilidade bem definida.

- **drawSubCategoryLabels:** esse método possui 88 linhas de código, Complexidade de *McCabe* igual a 14, possui seis parâmetros e possui Profundidade de Blocos Aninhados igual a quatro. Esse método realiza tarefas associadas a desenho gráfico. Ele possui uma série de *ifs*, que verificam se a borda trabalhada é do tipo *TOP*, *BOTTOM*, *LEFT* ou *RIGHT*, que poderiam ser substituídos por um *switch-case*, o que o tornaria melhor em termos de qualidade de código. Mas o que chama mais a atenção nesse método é que dentro desses quatro *ifs* há código duplicado devido à utilização de diferentes parâmetros em métodos que são chamados. Esse código duplicado poderia ser exportado para um novo método que recebesse esses parâmetros conforme o tipo da borda. No fim do método, essa situação se repete com outros quatro *ifs*. Esses dois trechos de quatro *ifs* tratam diferentes aspectos relacionados ao desenho que o método é responsável, podendo também ser exportados para novos métodos com a responsabilidade mais bem definida e mais simples.

- **findLocalMethods:** esse método possui 240 linhas de código, Complexidade de *McCabe* igual a 86, possui 18 parâmetros e possui Profundidade de Blocos Aninhados igual a 7. O método possui uma grande quantidade de parâmetros relacionados e que poderiam ser exportados para uma classe, além de ser grande e complexo, fazendo, inclusive, uso de *gotos*. Esses fatores prejudicam a sua legibilidade e compreensão.

- **prepMinion:** esse método possui 287 linhas de código, Complexidade de *McCabe* igual a 50, possui cinco parâmetros e possui Profundidade de Blocos Aninhados igual a 7. Percorrendo o método, percebe-se que ele é responsável por realizar muitas preparações (chamadas de "Minion") relacionadas ao comando de conexão com banco de dados. Esse método é demasiadamente longo e possui muitos comentários, sendo que o maior deles possui mais de 30 linhas. Esse tipo de situação corresponde ao *bad smell Comments* [Lanza et al., 2007], no qual tenta-se conduzir uma compensação da dificuldade de entendimento do código com comentários excessivos. Não é possível estabelecer o objetivo do método, seu nome não é claro e ele possui muitas responsabilidades.

- **readAj5ClassAttributes:** esse método possui 102 linhas de código, Complexidade de *McCabe* igual a 35, possui seis parâmetros e possui Profundidade de Blocos Aninhados igual a seis. Esse método possui quatro grandes blocos de código-fonte, que poderiam ser exportados individualizados para métodos que atendam a um único objetivo e auxiliem o entendimento futuro. Existem outros

dois métodos na classe semelhantes a esse método: `readAjd5MethodAttributes` e `readAj5FieldAttributes`. Isso mostra que a classe `AtAjAttributes` poderia ser sub-dividida em três outras classes que tratassem atributos do tipo *class*, *method* e *field*. Apesar disso não ter relação direta com o nível de método, uma observação que emerge dos resultados obtidos desse e de outros métodos inspecionados é que uma classe que trata de mais de uma responsabilidade tende a ter métodos mais complexos.

- **REPTree.Tree#buildTree:** esse método possui 137 linhas de código, Complexidade de *McCabe* igual a 29, possui 10 parâmetros e possui Profundidade de Blocos Aninhados igual a quatro. Da análise qualitativa, percebe-se que é um método recursivo que gera uma estrutura de dados no formato árvore. Ele possui diversos pontos de retorno, o que dificulta seu entendimento e é uma má prática de programação. Pelos comentários do próprio código, percebe-se que esse método desempenha várias tarefas, sendo esses trechos, inclusive, separados por mais de um espaço. O ideal seria que cada um desses trechos fosse exportado para um método que realizasse uma única tarefa.

- **startFileInternal:** esse método possui 90 linhas de código, Complexidade de *McCabe* igual a 21, possui nove parâmetros e possui Profundidade de Blocos Aninhados igual a quatro. Apesar do que o nome do método sugere, percebe-se que ele realiza muito mais do que inicializar um arquivo internamente, embora as tarefas aparentemente tenham relação entre si. O ideal seria que as responsabilidades fossem extraídas para métodos privados afim de facilitar o entendimento do código e reduzir sua complexidade.

- **visitBranchInstruction:** esse método possui 31 linhas de código, Complexidade de *McCabe* igual a cinco, possui cinco parâmetros e possui Profundidade de Blocos Aninhados igual a quatro. Esse método está exatamente dentro do limite inferior dos valores referência das quatro métricas. Por tal motivo, existe uma diferença sutil entre ele ser avaliado como *Ruim/Raro* pelos valores referência ou avaliados como *Regular/Ocasional*. A inspeção manual revelou um método relativamente compreensível, porém, com um interesse fora de seu contexto, que é o registro de *log*, dentro de um *if* que verifica se a aplicação está em modo de *DEBUG*. O objetivo do método não é registro de *log*, que é um interesse comumente espalhado pela aplicação. Uma forma correta de proceder com o registro de *log* seria o acionamento de uma classe cujo objetivo específico seja esse. Se esse *if* fosse removido do código por meio do acionamento de um método específico

de uma classe responsável por registro de *log*, as métricas MLOC, NBD e VG o teriam classificado na faixa *Regular/Ocasional*. Em relação aos parâmetros, percebe-se que três dos cinco são utilizados em todos os métodos da classe. Isso sugere uma relação entre esses parâmetros que pode vir a constituir uma nova classe do sistema.

9.1.2 Resultado da Inspeção de Classes

O Qualitas.*class* Corpus contém 247.395 classes. Foram combinadas todas as 10 métricas com valores referência derivados para nível de classe na classificação *Ruim/Raro* e não houve classes que atendessem a esses requisitos. Portanto, não há classes na amostra na qual todas as métricas correspondam às faixas *Ruim/Raro* de seus valores referências. Assim, a métrica Número de Métodos Estáticos foi excluída da análise, por ser menos relevante quando comparadas às demais. O filtro continuou muito restritivo, com apenas 1 classe sendo retornada. Posteriormente, removeu-se também o filtro da métrica Número de Filhas, haja vista que ela não tem impacto direto na qualidade da classe, e sim no risco de alteração e falha do projeto da hierarquia de herança.

Dadas essas duas exclusões, chegou-se a um número total de 19 classes, considerado viável para inspeção manual. Foram excluídas do resultado 3 classes: 2 delas são classes anônimas, que foram definidas mais de uma vez dentro das respectivas classes em que o recurso é utilizado, e, por tal motivo, atendeu à restrição *HAVING* $COUNT(*) = 8$ sem que, na verdade, tivessem as 8 métricas utilizadas na filtragem classificadas na faixa *Ruim/Raro* dentro de uma mesma definição anônima; 1 delas é uma classe *inner* que é definida dentro de vários métodos da classe *outer*, carregando o mesmo problema das classes anônimas, ou seja, apesar de atender a restrição da consulta SQL, nenhuma definição apresenta o critério especificado. As classes podem ser obtidas por meio do seguinte comando SQL no banco de dados mostrado na Seção 4.1.2.1.

```
1 SELECT
2   name, source, package, source_id
3 FROM
4 ( (SELECT name, source, package, source_id FROM measure
      where metric_id = 'WMC' AND value > 34)
5  UNION ALL (SELECT name,source,package, source_id FROM
       measure where metric_id = 'DIT' AND value > 4)
6  UNION ALL (SELECT name, source, package, source_id FROM
       measure where metric_id = 'LCOM' AND value > 0.725)
```

```
7  UNION ALL (SELECT name, source, package, source_id FROM
       measure where metric_id = 'NOM' AND value > 14)
8  UNION ALL (SELECT name, source, package, source_id FROM
       measure where metric_id = 'NOF' AND value > 8)
9  UNION ALL (SELECT name, source, package, source_id FROM
       measure where metric_id = 'NORM' AND value > 4)
10 UNION ALL (SELECT name, source, package, source_id FROM
       measure where metric_id = 'SIX' AND value > 1.333)
11 UNION ALL (SELECT name, source, package, source_id FROM
       measure where metric_id = 'NSF' AND value > 5)
12 ) as identified_uncommon_methods
13 GROUP BY name, source, package, source_id
14 HAVING count(*) = 8
15 ORDER BY name;
```

A Tabela 9.2 exibe as classes selecionadas, mostrando o nome, pacote e sistema. A seguir é descrita a avaliação qualitativa de cada uma dessas classes.

Tabela 9.2: Classes inspecionadas manualmente.

Classe	Pacote	Sistema
CloneableEditor	org.openide.text	netbeans-7.3
CompilationUnitEditor	org.eclipse.jdt.internal.ui.javaeditor	eclipse-3.7.1
CtxHelpTreeSection	org.eclipse.pde.internal.ua.ui.editor.ctxhelp	eclipse-3.7.1
DependencyManagementSection	org.eclipse.pde.internal.ui.editor.plugin	eclipse-3.7.1
DeploymentDisplay	megamek.client.ui.swing	megamek-0.35.18
GTKFileChooserUI	com.sun.java.swing.plaf.gtk	jre-1.6.0
InfoProduct	org.compiere.apps.search	compiere-330
JarPackageWizardPage	org.eclipse.jdt.internal.ui.jarpackager	eclipse-3.7.1
JoinNode	org.apache.derby.impl.sql.compile	derby-10.9.1.0
JRCTXVisualView	com.jaspersoft.ireport.designer.jrctx	iReport-3.7.5
JSVGCanvas	org.apache.batik.swing	batik-1.7
JTitledPanel	org.netbeans.lib.profiler.ui.components	netbeans-7.3
SAX2DTM2	com.sun.org.apache.xml.internal.dtm.ref.sax2dtm	jre-1.6.0
SAX2DTM2	org.apache.xml.dtm.ref.sax2dtm	xalan-2.7.1
SimpleCSMasterTreeSection	org.eclipse.pde.internal.ua.ui.editor.cheatsheet.simple	eclipse-3.7.1
SVGFlowRootElementBridge	org.apache.batik.bridge.svg12	batik-1.7

- **CloneableEditor:** a classe não possui um objetivo bem definido. Há trechos de código de *log*, de manipulação de arquivos e de manipulação de componentes gráficos. Trata a interação com o usuário e uma série de manipulações, tendo várias responsabilidades. A classe tem muitas tarefas, indicando a necessidade de refatoração para classes mais específicas, que sejam mais fáceis de entender.

- **CompilationUnitEditor:** essa classe possui uma Profundidade de Árvore de Herança $DIT = 8$, sendo uma tarefa árdua tentar entender a hierarquia, afinal, o projeto se torna mais complexo. Além disso, a classe é muito grande e é difícil descrever um papel bem definido para ela. Pela hierarquia de herança profunda observada, ela deveria ser bastante específica, o que não ocorre de fato. Ou seja, além da complexidade inerente ao tamanho da própria classe, ela herda mais métodos e se torna ainda mais complexa. Isso sugere fortemente que a hierarquia de herança pode estar equivocada. Há tantos métodos na classe que é difícil estabelecer um propósito específico, o que está de acordo com o déficit de coesão indicado pelo valor referência. A classe realmente parece realizar muitas tarefas, sendo difícil compreendê-la, o que pode tornar difícil o reúso e a manutenção dela.

- **CtxHelpTreeSection:** essa classe possui uma Profundidade de Árvore de Herança $DIT = 6$, sendo uma hierarquia de herança complexa. Possui um conjunto de constantes inter-relacionadas que poderiam estar mais bem colocadas dentro do projeto em uma estrutura de enumeração (*enum*). Apesar de a classe possuir métodos em sua maioria concisos, percebe-se que falta relação entre eles, podendo ser reagrupados em classes mais coesas, com papéis mais específicos e que descrevam melhor o projeto e facilitem o entendimento e a manutenção. Assim sendo, a classe é candidata à refatoração, pois ela parece estar realizando tarefas que deveriam ser realizadas por mais de uma classe.

- **DependencyManagementSection:** é uma classe que possui alto valor de DIT e que, além de implementar muitas tarefas, possui muitos métodos, o que resulta em uma classe complexa. Também possui uma série de constantes que estariam mais bem colocadas em uma *enum*. Percebe-se que a classe trata uma série de papéis que poderiam estar mais bem colocados em classes mais coesas, o que tornaria o programa mais legível e possivelmente a manutenção mais fácil.

- **DeploymentDisplay:** exibe um componente gráfico Swing. Porém, percebe-se que, além da tela em questão poder ser mais bem componentizada, há código que deveria estar em uma camada de controle, supondo um padrão arquitetural MVC. Um exemplo de melhor componentização seria, por exemplo, o *JButton* de *back* e *forward* da classe estarem definidos em um componente específico de navegação. Outro ponto importante nessa classe são seis métodos que mostram código duplicado, que poderiam estar em um único método com os devidos parâmetros.

- **GTKFileChooserUI:** é uma classe muito grande, que poderia ser dividida em mais classes. Um ponto que chama a atenção é a quantidade de atributos. Esses atributos, são, inclusive, mostrados em sub-grupos mais coesos dentro do código da classe. Não existe espaço entre os grupos mais coesos de atributos, e, quando muda-se o grupo, há uma série de quebras de linha que sugerem um novo grupo de atributos mais relacionados. O primeiro grupo de atributos estão nomeados no padrão *newFolder*%. Ou seja, há quatro atributos relacionados à criação de uma nova pasta, que poderiam estar em uma classe referente a uma ação de criação de nova pasta, juntamente com os comportamentos associados à essa ação. O próprio programador é consciente de uma relação mais coesa entre os atributos declarados, tentando transparecer isso por meio de um agrupamento na própria classe, que não é o ideal para aspectos de qualidade de software. Como resultado, a classe é grande, complexa e de difícil compreensão de objetivos específicos, realizando muitas tarefas. Na classe há, inclusive, métodos vazios. É uma classe de interface com o usuário mal componentizada.

- **InfoProduct:** essa classe contraria os princípios básicos de arquitetura de software, pois contém código referente a visão, controle e acesso a banco de dados, inclusive com atributos que são *strings* com comandos SQL, tudo na mesma classe. É uma classe que contém configurações de tela e não a componentiza bem dentro da camada de visão.

- **JarPackageWizardPage:** é um componente de visualização que representa a primeira página da opção de exportação de *JAR* no *Eclipse*. Tem muitos atributos, principalmente do tipo *Button*. Percebe-se um padrão de qualidade deteriorada nas classes do *Eclipse* que são responsáveis pelas telas, visivelmente mal componentizadas. Por exemplo, nessa classe, há atributos relacionados a opções de exportação e tratamentos desses dados que poderia estar componentizados em uma outra classe. Pela falta de componentização, essas classes acabam se tornando complexas, com muitos atributos, muitas constantes e comportamentos especializados. Como consequência dessa complexidade, a classe não é coesa em termos qualitativos nem em termos quantitativos.

- **JoinNode:** o próprio comentário da classe sugere que ela não é coesa. O comentário diz que ela representa um resultado de junção para *SELECT* e *INSERT - SELECT* quando de operações do tipo de manipulação de dados. Percebe-se, então, que, mesmo no nível de projeto, a classe tem duas responsabilidades. Em termos de código, percebe-se uma classe longa e complexa, com muitos atributos

e muitas constantes. As constantes indicam o tipo de junção, devendo estar em uma *enum*. Essa classe também parece ter o *bad smell Comments*, pois apresenta comentários desproporcionais aos trechos comentados, indicando uma falta de legibilidade que tenta ser compensada com um comentário extenso. Com isso, avalia-se que a classe tem baixa coesão, é complexa e deve ser candidata à refatoração.

- **JRCTXVisualView:** é uma classe associada a uma tela do sistema. Possui uma classe de *controller* associada, e o que aparenta ser alguns atributos que são subcomponentes da tela em questão. A classe é extensa e é difícil estabelecer um propósito específico sobre o tipo de tela que ela trata. O comentário da classe não traz informação significativa: "*Top component which displays something*". A classe tem muitos atributos e métodos que são tratados de forma isolada, indicando baixa coesão. Dessa forma, a classe deveria ser analisada como candidata à refatoração.

- **JSVGCanvas:** é uma classe aparentemente bem programada, com linhas curtas que permitem uma boa visualização e comentários concisos. Seus métodos também são curtos. Porém, é uma classe grande e com um propósito geral de definição de um componente *Swing*, segundo o comentário exposto para a classe. Não há componentização suficiente para que a classe apresente clareza e singularidade dentro do projeto. Tanto as ações como ítens da tela poderiam estar componentizados em classes menos complexas e com propósito mais bem definido.

- **JTitledPanel:** é uma classe de visão, que expõe um painel relacionado a título ou a região superior de uma tela. Essa classe tem três classes internas que prejudicam a sua avaliação. Possui, visivelmente, necessidade de melhorias, com o particionamento das responsabilidades em outras classes que possuam melhor coesão. Por exemplo, existem dois *JButtons* relacionadas a minimização e maximização, com métodos respectivos a manipulação de ações sobre eles. Essa responsabilidade poderia ser exportada para uma nova classe, que possuísse coesão e singularidade em relação a esse tipo de ação. Apesar disso, a classe mostra métodos enxutos e aparentemente bem definidos, carecendo apenas de uma melhor definição de responsabilidades singulares para as classes.

- **SAX2DTM2 (jre-1.6.0/xalan-2.7.1):** essa classe implementa um DTM (*Document Table Model*) para conversão de XML para uma estrutura de tabela, utilizando SAX2 (*Simple API for XML*). Essa classe parece ter o *bad smell comments*, pois os comentários não são concisos. Há mais de um atributo com comentário

referente de mais de 10 linhas, bem como linhas de código de métodos comentadas de forma excessiva. A classe expõem muitos métodos, que mostram muitos laços e estruturas de controle. A classe possui alguns métodos vazios, sem qualquer implementação, publicamente expostos. Apesar de um propósito aparentemente bem definido, a prática não mostra que a classe é coesa, devendo ser candidata à refatoração, por meio da exportação de atributos e métodos para outras classes que realizem especificidades referentes à conversão. Percebe-se também código referente a *log* na classe, que deveria estar implementado em uma classe específica. Percebe-se também código duplicado nessa classe. Ela estende SAX2DTM, por implementar otimizações relacionadas à conversão, sobrescrevendo operações que podem ser executadas de forma mais eficiente. No próprio comentário sobre a classe, há uma referência de que a extensibilidade da classe é limitada pelo fato de haver manipulação de aspectos diretos relacionados ao desempenho, que podem impactar na sua qualidade interna e, consequentemente, em aspectos de qualidade externa como a extensibilidade. Além disso, os comentários na classe expõem a fragilidade dela, pois sugerem que tenha bastante cuidado ao alterá-la. Tudo isso indica que se trata de uma classe difícil de modificar, entender, reusar, com baixos atributos de qualidade interna que podem impactar diretamente em seus atributos de qualidade externa.

- **SimpleCSMasterTreeSection:** ao inicializar a inspeção da classe, percebe-se uma lista muito grande de constantes que prejudica a avaliação. Elas poderiam estar mais bem agrupadas em um *enum*, cujo nome, inclusive, poderia informar melhor sobre sua utilização. Essa classe possui atributos do tipo %Action, que indicam que as ações são separadas em objetos que as tratam de forma mais singular, o que não ocorreu na outra classe analisada do mesmo software: `DependencyManagementSection`. Embora isso tenha ocorrido, a classe também trata diversas ações que poderiam estar agrupadas em outras classes.

- **SVGFlowRootElementBridge:** possui muitas constantes, que deveriam estar em uma estrutura do tipo *enum*, melhor modularizando o código e o tornando mais legível. É uma classe com muitos métodos, muitos deles sobrescritos, alguns muito longos e de propósitos diferentes. Os métodos poderiam ser mais bem agrupados em classes menores e mais coesas.

9.1.3 Discussão

9.1.3.1 Métodos

Os 10 métodos avalidos foram selecionados de forma aleatória dentre uma combinação dos 4 valores referência sugeridos que avaliam métodos. Esses métodos carregam uma grande complexidade, tamanho e parâmetros, tornando-os entidades problemáticas dentro do software, indo contra princípios básicos da programação orientada por objetos. Esses métodos possuem oportunidades claras de melhorias e a identificação delas não exige, nem mesmo, muito conhecimento acerca do domínio do software. Os resultados deste estudo mostram que os valores referência propostos são capazes de identificar métodos nessa situação.

9.1.3.2 Classes

As 16 classes avaliadas foram selecionadas por meio da combinação de 8 valores referência sugeridos, que avaliam diferentes aspectos relacionados à qualidade de software em classes de softwares orientados por objetos. Nessas classes, foi realizada uma inspeção manual com o objetivo de avaliá-las qualitativamente, que indicou que todas elas possuem problemas de qualidade interna, sendo pouco legíveis, pouco coesas, de difícil entendimento, o que as torna complexas, difíceis de reutilizar ou estender e de difícil modificação e manutenção. O resultado da inspeção qualitativa está de acordo com os valores referência utilizados na identificação quantitativa dessas classes. Essas classes podem ser um risco ao projeto, pois tornam as tarefas inerentes à manutenção e evolução de software mais complexas, devido à limitação de entendimento que acompanha esse tipo de classes.

Há estudos que mostram que profissionais da área de desenvolvimento de software gastam pelo menos de 30% a 50% do tempo entendendo o código para o qual irão realizar a manutenção [Paige & Meyer, 2008]. A capacidade de identificar essas classes segundo parâmetros quantitativos de qualidade interna, evitando as inspeções manuais, que podem ser caras, é uma oportunidade valiosa de melhorar a qualidade de software, tanto durante a fase de construção, como durante a fase de manutenção e evolução.

9.2 Estudo de Caso 3

Nesse estudo de caso, é analisado o software *JHotDraw*, que é um *framework* desenvolvido em Java para desenho técnico e gráfico. *JHotDraw* foi desenvolvido como um exercício de padrões de projeto em suas primeiras versões, mas é uma ferramenta gráfica

poderosa. Esse software foi escolhido pois seu projeto é considerado de alta qualidade [Riehle, 2000] e tem sido utilizado como tal em outros estudos de caso [Ferreira et al., 2012].

Partindo do pressuposto de que o software *JHotDraw* é bem projetado, o objetivo deste estudo de caso é verificar como os valores referência propostos se comportam na avaliação de um software bem projetado. Este estudo é complementar ao realizado no Capítulo 8 em direção à avaliação dos valores referência propostos em softwares bem e mal e projetados. Espera-se que a maioria das medidas do *JHotDraw* sejam bem avaliadas pelos valores referência de métricas de classes. Esse resultado sugeriria que os valores referência avaliam um software como bom quando ele realmente o é, o que permitiria chegar a duas conclusões. A primeira delas é que uma medida na faixa *Bom/Frequente* é confiável em relação à expectativa qualitativa acerca da qualidade de software e, consequentemente, minimiza-se o problema de avaliações *falso-positivas*, que é quando a classe não possui problemas estruturais, mas os valores referência os indicam. Nesse caso, espera-se que a maioria das classes não possuam problemas estruturais e os valores referência não os indiquem. A segunda delas é que avaliações *falso-negativas*, isto é, quando a classe possui problemas e o valor referência não os indicam, também serão minimizadas.

Avaliou-se o *JHotDraw* Versão 7.5.1, a versão compilável para o *Eclipse* disponibilizada no Qualitas.*class* Corpus. O conjunto de medidas avaliadas foram parte do *dataset* utilizado nas derivações. Segundo os dados coletados, tal versão possui 66 pacotes e 1.061 classes avaliadas por meio das métricas desse nível de análise. A Tabela 9.3 descreve a classificação obtida pelas classes do sistema por meio da aplicação dos valores referência propostos. A coluna **+1** apresenta a quantidade de classes de forma absoluta e relativa classificada na faixa *Bom/Frequente*, tal como a coluna **0** e **-1**, que apresentam os mesmos dados, porém para as faixas *Regular/Ocasional* e *Ruim/Raro*, respectivamente. A coluna **+1 OU 0** apresenta a quantidade absoluta e relativa de classes que não foram mal avaliadas pelos valores referência sugeridos.

Pela coluna **+1 OU 0**, percebe-se que 5 dos 10 valores referência propostos avaliam mais de 95% das classes com qualidade boa ou regular, sendo que nesses casos menos de 5% são mal avaliadas. O restante dos valores referência avalia pelo menos cerca de 85% das classes com qualidade boa ou regular.

Outro dado sugestivo da boa avaliação obtida pelas métricas do *JHotDraw* resultante dessa tabela é que, na média (coluna **+1 OU 0**), 91,56% das classes não são mal avaliadas.

Com o auxílio da *RAFTool*, descrita no Capítulo 7, foram realizados alguns testes em relação ao processo de composição dos valores referência para filtragem de classes,

Tabela 9.3: Contagem absoluta e relativa de classes classificadas em cada uma das faixas dos valores referência sugeridos e das classes que não foram mal avaliadas.

Métrica	+1	0	+1 OU 0	-1
Número de Atributos	892 (84,07%)	133 (12,54%)	1025 (96,61%)	36 (3,39%)
Número de Métodos	710 (66,92%)	226 (21,30%)	936 (88,22%)	125 (11,78%)
Métodos Ponderados por Classe	668 (62,96%)	273 (25,73%)	941 (88,69%)	120 (11,31%)
Profundidade de Árvore de Herança	715 (67,39%)	222 (20,92%)	937 (88,31%)	124 (11,69%)
Ausência de Coesão em Métodos	805 (75,87%)	154 (14,51%)	969 (90,39%)	102 (9,61%)
Número de Métodos Estáticos	990 (93,31%)	19 (1,79%)	109 (95,10%)	52 (4,90%)
Número de Atributos Estáticos	961 (90,57%)	54 (5,09%)	1015 (95,66%)	46 (4,34%)
Índice de Especialização	649 (61,17%)	252 (23,75%)	901 (84,92%)	160 (15,08%)
Número de Métodos sobrescritos	910 (85,77%)	78 (7,35%)	988 (93,12)	73 (6,88%)
Número de Filhas	969 (91,33%)	48 (4,52%)	1071 (95,55%)	44 (4,15%)

para verificar quantas classes seriam retornadas para a inspeção manual:

- ***UNCOMMON*[WMC] AND *UNCOMMON*[NOF] AND *UNCOMMON*[NOM] AND *UNCOMMON*[LCOM]:** essas métricas são comumente adotadas para a detecção de *God Classes* [Lanza et al., 2007], pois sugerem uma classe grande, complexa e de baixa coesão. Ao aplicar esse mecanismo de filtragem de classes no *JHotDraw* por meio da *RAFTool*, foram retornadas apenas 15 classes, ou seja, menos de 1,5% do total de classes do sistema.

- ***UNCOMMON*[NORM] AND *UNCOMMON*[DIT] AND *UNCOMMON*[SIX]:** são métricas que se complementam na avaliação da herança de uma classe. Quando se fala de uma classe mal avaliada pelo valor referência de DIT, NORM e SIX, esperam-se classes com uma árvore de herança profunda e, mesmo assim, que não reutilizam comportamentos das classes mãe, devido ao alto valor de NORM e que possui uma execução de comportamentos especializados em tempo de execução fora do normal, o que a torna mais difícil de ser testada. Ao aplicar esse mecanismo de filtragem de classes no *JHotDraw* por meio da *RAFTool*, foram retornadas apenas 10 classes, ou seja, menos de 1% do total de classes do sistema.

- ***UNCOMMON*[DIT] AND *UNCOMMON*[NSC]:** nesse caso, além de buscar classes com profundidade incomum de árvore de herança, foram buscadas classes com um número incomum de filhas. Esse tipo de classe, além de ser complexa devido à pronfundidade da árvore de herança, apresenta uma manutenção de alto risco, pois, como possuem um grande Número de Filhas, sua modificação pode ter impacto em um número considerável de outras classes. No entanto, ao aplicar esse filtro, apenas 2 classes foram retornadas, menos de 0,2% do total.

- ***UNCOMMON*[WMC] AND *UNCOMMON*[LCOM]:** esse mecanismo é simples e busca classes complexas e de baixa coesão. Foram retornadas 47 classes do *JHotDraw*, cerca de 4% das classes do sistema.

- ***UNCOMMON*[NOF] AND *UNCOMMON*[NOM] AND *UNCOMMON*[WMC] AND *UNCOMMON*[DIT] AND *UNCOMMON*[LCOM] AND *UNCOMMON*[SIX] AND *UNCOMMON*[NRM] AND *UNCOMMON*[NSC]:** esse mecanismo envolve todas as métricas, com exceção de NSM e NSF, que avaliam atributos e métodos estáticos. Não há classes que atendam a esse filtro no *JHotDraw*.

Considerando-se cada métrica isoladamente, têm-se um número relativamente baixo de classes mal avaliadas. Da mesma forma, a partir da combinação de métricas por meio de mecanismos de filtragem baseados na faixa *Ruim/Raro*, obteve-se um número relativo ainda menor de classes mal avaliadas.

9.2.1 Discussão

Para todas as métricas avaliadas, a maioria das classes do *JHotDraw* foram classificadas na faixa *Bom/Frequente*, o segundo maior grupo na faixa *Regular/Ocasional* e poucas classes na faixa *Ruim/Raro*. Tendo em vista as distribuições dos valores das métricas, é possível esperar esse tipo de resultado na maioria dos sofwares, porém, os números obtidos pelo *JHotDraw* são expressivos no direcionamento da indicação quantitativa de sua qualidade. As faixas *Bom/Frequente* e *Regular/Ocasional* não foram avaliadas individualmente porque é difícil diferenciar com precisão se uma classe é de qualidade boa ou regular.

Conforme observado no trabalho de Ferreira et al. [2012], a faixa *Regular/Ocasional* pode ser considerada como uma recomendação de que a estrutura pode ser melhorada, embora não represente necessariamente uma ocorrência de anomalia. Portanto, problemas relacionados à fronteira estabelecida para as faixas *Bom/Frequente* e *Regular/Ocasional* não tem consequências tão representativas para a avaliação quantitativa do software. Entretanto, a fronteira entre as faixas *Regular/Ocasional* e *Ruim/Raro* é, de fato, sutil. No entanto, há de se lembrar que os valores aqui estabelecidos são valores referência, e não dispensam a análise de um especialista acerca dos resultados obtidos. Outra consideração importante é que, conforme observado no Capítulo 8, a avaliação do método, classe ou pacote por um único valor referência não é recomendada na avaliação da qualidade, pois uma avaliação mais ampla proporciona resultados quantitativos de maior amplitude de acurácia. Um dos recursos desenvolvidos na ferramenta *RAFTool*

é a possibilidade de estabelecer um critério de ordenação de forma que, ao se optar, por exemplo, pelas classes classificadas por Número de Atributos como *Regular/Ocasional*, o especialista possa ter ciência de qual é a classe que, mesmo não sendo mal avaliada, mostra o maior valor da métrica. Isso auxilia a tomada de decisão quando da utilização dos valores referência.

9.3 Limitações

A principal limitação do Estudo de Caso 2 é relacionada à falta de conhecimento do domínio do problema dos métodos e classes inspecionados, bem como um conhecimento mais geral acerca do projeto do software avaliado. A principal limitação do Estudo de Caso 3 está relacionada à premissa formulada com base em outros estudos de que o software *JHotDraw* é de alta qualidade. Essa premissa depende da validade desses estudos, embora os números reflitam o panorama de qualidade sugerido qualitativamente.

9.4 Conclusão

Os resultados obtidos no Estudo 1 indicam que uma medição avaliada na faixa *Ruim/-Raro* pelos valores referência sugeridos é confiável em relação à expectativa acerca da qualidade deteriorada daquela entidade. Os resultados do estudo mostram que avaliações *falso-positivas* são mínimas quando os valores referência são aplicados, uma vez que se observou uma forte associação entre os resultados obtidos quantitativa e qualitativamente.

Os resultados obtidos no Estudo 2 sugerem que uma medição na faixa *Bom/-Frequente* é confiável, ou seja, os valores referência são capazes de detectar, de forma adequada, a boa qualidade estrutural de uma entidade. Como consequência, conclui-se que, com a aplicação dos valores referência propostos, avaliações *falso-negativas* são minimizadas. Os resultados também sugerem que avaliações *falso-positivas*, isto é, quando as classes não possuem problemas e os valores referência os indicam, também serão minimizadas. Isso ocorre pois, devido ao consenso qualitativo da boa qualidade acerca do *JHotDraw*, era esperado que a maioria das classes não possuíssem problemas e que isso fosse refletido por meio da aplicação dos valores referência, o que foi confirmado pelo estudo de caso. Ou seja, os valores referência não apontaram problemas em classes bem construídas.

Portanto, os valores referência avaliados por meio desses estudos de casos sugerem que eles podem ser aplicados para avaliarem a qualidade de métodos e classes dentro

de softwares orientados por objetos. A aplicação desses valores referência na prática podem prover um meio de menor custo do que a condução de inspeções manuais para a identificação de métodos e classes problemáticos dentro de um panorama de controle da qualidade de software. No entanto, é importante ressaltar que a inspeção manual conduzida por um engenheiro de software continua indispensável dentro do gerenciamento da qualidade, uma vez que um artefato mal avaliado quantitativamente deve ser objeto de análise. Logo, os valores referência proporcionam uma forma de as avaliações quantitativa e qualitativa poderem se complementar de forma a conduzir uma avaliação qualitativa do software mais eficiente.

Capítulo 10

Conclusão

Uma visão quantitativa dos atributos de qualidade interna de software orientado por objetos é de fundamental importância para a construção de princípios sólidos na Engenharia de Software. Nesse contexto, as métricas de software apresentam-se como o meio de medir esses atributos, quantificando a qualidade de aspectos essenciais a um projeto de software, relacionados a sua qualidade interna. Da mesma forma, valores referência que possam sugerir desvios na qualidade interna do projeto aos profissionais da Engenharia de Software são importantes durante as fases de modelagem, desenvolvimento e operação, de forma a evitar problemas futuros.

Valores referência podem viabilizar o uso de métricas pelo engenheiro de software e otimizar a alocação de recursos em pontos que podem ser fundamentais para o sucesso do projeto de software. Por exemplo, uma classe do sistema que apresenta um valor alto para LCOM, dado o valor referência estabelecido, pode ser refatorada, melhorando a qualidade interna do software e, consequentemente, diminuindo sua propensão a *bugs*. Isso tem impacto direto na qualidade externa do software, trazendo para o engenheiro de software um poder de avaliação fundamental. Da mesma forma, classes mal avaliadas podem receber uma atenção maior durante a fase de testes, otimizando o processo de manutenção e gerando um software com maior grau de corretude.

Nesta dissertação de mestrado foi proposto um catálogo com valores referência para 18 métricas de softwares orientados por objetos, que cobrem não somente a avaliação de classes, mas também de métodos e pacotes. Além disso, foram identificadas as distribuições que possuem melhor encaixe com o conjunto de medidas dessas métricas. Utilizou-se o método empírico proposto por Ferreira et al. [2012], que é baseado na análise da distribuição estatística de medidas encontradas na prática. Neste trabalho foram sugeridas e adotadas algumas melhorias no método que avançaram nas definições e técnicas estatísticas utilizadas, bem como ampliou-se a discussão acerca dos aspec-

tos positivos e negativos que envolvem seus resultados. Foi observado que o conjunto de medidas das métricas se encaixa com distribuições cauda-pesada ou inclinadas à direita, nas quais a média não é representativa. Dessa forma, três faixas de valores foram tomadas para identificação dos valores referência propostos: *Bom/Frequente*, *Regular/Ocasional* e *Ruim/Raro*. Apesar de esses valores não expressarem necessariamente as melhores práticas da Engenharia de Software, eles refletem o padrão de qualidade seguido pela maioria dos softwares. Os valores referência foram avaliados por meio de um experimento e três estudos de caso. Os resultados desses estudos indicam que os valores referência estabelecidos têm a real capacidade de indicar o panorama de qualidade de métodos, classes e pacotes por meios quantitativos.

A ferramenta *RAFTool*, desenvolvida como parte deste trabalho de dissertação, suporta a realização da filtragem de métodos, classes e pacotes em risco no projeto de software, baseada nos valores referência propostos. A entrada dessa ferramenta é um arquivo *XML* contendo as medições do sistema, obtido facilmente por meio do *plugin Metrics* do Eclipse. Essa ferramenta proporciona uma forma simples e eficiente de aplicar os valores referência na filtragem de métodos, classes e pacotes baseado em indicativos quantitativos do software, suportando a identificação de medições anômalas dentro do contexto do gerenciamento da qualidade interna dos softwares.

Portanto, com o catálogo de valores referência proposto e outras contribuições secundárias realizadas neste trabalho de dissertação, contribui-se com os esforços de pesquisa na promoção das métricas de software como um instrumento efetivo no gerenciamento da qualidade interna de softwares.

Como trabalhos futuros, é importante evoluir a ferramenta *RAFTool* de forma a integrá-la efetivamente no ambiente de desenvolvimento de software, criando um *plugin* para o *Eclipse* que seja capaz de identificar medições anômalas em tempo de construção e manutenção do software, disparando "alarmes" que indiquem a ameaça à qualidade do código-fonte ao programador e sugiram possíveis refatorações que possam ser aplicados com o objetivo de melhorar o código.

Uma necessidade de continuidade de trabalho que emergiu das observações do experimento realizado é a proposição de um método quantitativo para sugerir transferências de classes para pacotes mais apropriados, bem como a criação de novos subpacotes que melhor agrupem um conjunto de classes que estão distribuídas em outros pacotes.

Faz-se também necessária a utilização dos valores referência sugeridos como base para a identificação de *bad smells* e *anti-patterns*, avançando não somente na área de detecção de *bad smells*, mas também ampliando a avaliação dos valores referência em cenários reais da Engenharia de Software.

Outro trabalho futuro importante está relacionado à visualização de software. As abordagens atuais contruídas com a utilização de métricas consideram medidas absolutas. Por exemplo, na visão polimétrica, retângulos mostram as entidades de software e sua altura e largura mostram duas métricas, por exemplo, LOC e WMC em seus valores absolutos. Porém, uma visualização que mostre a faixa na qual a entidade foi avaliada por meio de um tamanho fixo pode ser aplicada para melhorar a visualização de software baseada em métricas por meio dos valores referência.

Capítulo 11

Publicações

Esta dissertação deu origem às seguintes publicações:

1. *Um Método de Extração de Valores Referência para Métricas de Softwares Orientados por Objetos.* In: Congresso Brasileiro de Software: Teoria e Prática, 2014, Maceió/AL. IV Workshop de Teses e Dissertações do CBSOFT, 2014. v. 2. p. 62-67.

2. *Statistical dataset on software metrics in object-oriented systems.* Software Engineering Notes, v. 39, p. 1-6, 2014.

3. *RAFTool - Ferramenta de Filtragem de Métodos, Classes e Pacotes com Medições Incomuns de Métricas de Software.* In: WAMPS 2014 - Trilha de Ferramentas.

4. *A Catalogue of Thresholds for Object-Oriented Software Metrics*, submetido para uma conferência internacional.

Referências Bibliográficas

Alshayeb, M. (2009). Refactoring effect on cohesion metrics. Em *Computing, Engineering and Information, 2009. ICC '09. International Conference on*, pp. 3–7.

Alves, T. L.; Ypma, C. & Visser, J. (2010). Deriving metric thresholds from benchmark data. Em *ICSM*, pp. 1–10. IEEE Computer Society.

Anquetil, N. & Laval, J. (2011). Legacy software restructuring: Analyzing a concrete case. Em *Software Maintenance and Reengineering (CSMR), 2011 15th European Conference on*, pp. 279–286. ISSN 1534-5351.

Baxter, G.; Frean, M.; Noble, J.; Rickerby, M.; Smith, H.; Visser, M.; Melton, H. & Tempero, E. (2006). Understanding the shape of java software. Em *OOPSLA '06: Proceedings of the 21st annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*, pp. 397--412, New York, NY, USA. ACM.

Beirlant, J.; Matthys, G. & Dierckx, G. (2001). Heavy-tailed distributions and rating. *ASTIN Bulletin*, pp. 37--58.

Benlarbi, S.; Emam, K. E.; Goel, N. & Rai, S. (2000). Thresholds for Object-Oriented Measures. Em *Proceedings of the 11th International Symposium on Software Reliability Engineering*, ISSRE '00, pp. 24----, Washington, DC, USA. IEEE Computer Society.

Bloch, J. (2008). *Effective Java (2Nd Edition) (The Java Series)*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2 edição. ISBN 0321356683, 9780321356680.

Bouwers, E.; Visser, J. & van Deursen, A. (2012). Getting what you measure. *Commun. ACM*, 55(7):54--59. ISSN 0001-0782.

Brito e Abreu, F. & Carapuca, R. (1994). Object-oriented software engineering: Measuring and controlling the development process. Em *Proc. Int'l Conf. Software Quality (QSIC)*.

Chhikara, A.; Chhillar, R. S. & Khatri, S. (2011). Evaluating the impact of different types of inheritance on the object oriented software metrics. *Internation Journal of Enterprise Computing and Business Systems*, 1(2). ISSN 22308849-V1I2M7-072011.

Chidamber, S. R. & Kemerer, C. F. (1994). A metrics suite for object oriented design. *IEEE Trans. Softw. Eng.*, 20(6):476--493. ISSN 0098-5589.

Cunningham, W. (1992). The wycash portfolio management system. *SIGPLAN OOPS Mess.*, 4(2):29--30. ISSN 1055-6400.

Doane, D. & Seward, L. (2011). Measuring Skewness: A Forgotten Statistic? *Journal of Statistics Education*, 85(2):1--10.

EasyFit (2013). Easyfit. Disponível em: http://www.mathwave.com/products/easyfit.html. Acesso em Setembro de 2013.

Eclipse (2013). Eclipse. Disponível em: http://www.eclipse.org. Acesso em Setembro de 2013.

Ferreira, K. A.; Bigonha, M. A.; Bigonha, R. S.; Mendes, L. F. & Almeida, H. C. (2012). Identifying thresholds for object-oriented software metrics. *Journal of Systems and Software*, 85(2):244--257. ISSN 01641212.

Ferreira, K. A. M. (2006). Avaliação de Conectividade em Sistemas Orientados por Objetos. Dissertação de mestrado, DCC/UFMG.

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.; Almeida, H. C. & Neves, R. C. (2011). Métrica de Coesao de Responsabilidade - A Utilidade de Métrica de Coesão de Responsabilidade na Identificação de Classes com Problemas Estruturais. Em *In: X Simpósio Brasileiro de Qualidade de Software*, pp. 1--10. Proceedings of X Simpósio Brasileiro de Qualidade de Software.

Foss, S.; Korshunov, D. & Zachary, S. (2009). An introduction to heavy-tailed and subexponential distributions.

Fowler, M. (1999). *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, Boston, MA, USA. ISBN 0-201-48567-2.

Fowler, M. (2003). Anemic domain model. Disponível em: http://www.martinfowler.com/bliki/AnemicDomainModel.html. Acesso em Setembro de 2013.

Gibbons, J. D. & Chakraborti, S. (2003). *Nonparametric Statistical Inference (Statistics: a Series of Textbooks and Monogrphs)*. CRC, 4 edição. ISBN 0824740521.

Hevery, M. (2008). Google testing: Static methods are death to testability. Disponível em: http://googletesting.blogspot.com.br/2008/12/static-methods-are-death-to-testability.html. Acesso em Fevereiro de 2014.

Kaur, S.; Singh, S. & Kaur, H. (2013). A Quantitative Investigation Of Software Metrics Threshold Values At Acceptable Risk Level. *International Journal of Engineering Research & Technology (IJERT)*, 2(3):1--7.

Lamrani, M.; Amrani, Y. E. & Ettouhami, A. (2011). Formal Specification of Software Design Metrics. Em *The Sixth Internation Conference on Software Engineering Advances*, pp. 348--355.

Lamrani, M.; Amrani, Y. E. L. & Ettouhami, A. (2013). A formal definition of metrics for object oriented design: mood metrics. *Journal of Theoretical and Applied Information Technology*, 49(1):1--10.

Lanza, M.; Ducasse, S. & Marinescu, R. (2007). *Object-Oriented Metrics in Practice: Using Software Metrics to Characterize, Evaluate, and Improve the Design of Object-Oriented Systems*. Springer. ISBN 9783540395386.

Lehman, M. M. (1978). Programs, cities, students, limits to growth? *Programming Methodology*, pp. 42–62. Inaugural Lecture.

Lehman, M. M. (1980). Programs, life cycles, and laws of software evolution. *Proc. IEEE*, 68(9):1060--1076.

Li, W. (1998). Another metric suite for object-oriented programming. *Journal of Systems and Software*, 44(2):155--162. ISSN 0164-1212.

Lopez, M. & Habra, N. (2005). Relevance of the cyclomatic complexity threshold for the java programming language. Em Dekkers, T., editor, *Proceedings of the 2nd Software Measurement European Forum (SMEF2005)*, pp. 195--202.

Lorenz, M. & Kidd, J. (1994). *Object-Oriented Software Metrics*. Prentice Hall. ISBN 013179292X.

Malik, N. & Chhillar, R. S. (2011). New Design Metrics for Complexity Estimation in Object Oriented Systems. 3(10):3367--3382.

Martin, R. (1994). OO design quality metrics - an analysis of dependencies. Em *Workshop Pragmatic and Theoretical Directions in Object-Oriented Software Metrics*. OOPSLA'94.

Matlab (2014). Matlab. Disponível em: http://www.mathworks.com/help/stats. Acesso em Janeiro de 2014.

McCabe, T. (1976). A complexity measure. *Software Engineering, IEEE Transactions on*, SE-2(4):308–320. ISSN 0098-5589.

Metrics, E. (2014). Eclipse metrics. Disponível em: http://metrics2.sourceforge.net/. Acesso em Fevereiro de 2014.

Milojević, S. (2010). Power law distributions in information science: Making the case for logarithmic binning. *Journal of the American Society for Information Science and Technology*, 61(12):2417--2425. ISSN 1532-2890.

Mishra, D. (2011). New Inheritance Complexity Metrics for Object-Oriented Software Systems: An Evaluation with Weyuker's Properties. *Computing & Informatics*, 30(2):267--293. ISSN 13359150.

Newman, M. E. J. (2003). The Structure and Function of Complex Networks. *SIAM Review*, 45(2):167--256.

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.

Oracle (2014a). Core j2ee patterns - data access object. Disponível em: http://www.oracle.com/technetwork/java/dataaccessobject-138824.html. Acesso em Junho de 2014.

Oracle (2014b). The java ee 6 tutorial. Disponível em: http://docs.oracle.com/javaee /6/tutorial/doc/bnaqm.html. Acesso em Junho de 2014.

Oracle (2014c). Java se technical documentation. Disponível em: http://docs.orac le.com/javase/. Acesso em Fevereiro de 2014.

Paige, R. F. & Meyer, B. (2008). *Objects, Components, Models and Patterns: 46th International Conference, TOOLS EUROPE 2008, Zurich, Switzerland, June 30-July 4, 2008, Proceedings*. Springer Publishing Company, Incorporated, 1 edição. ISBN 354069823X, 9783540698234.

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

R (2014a). Chi-squared test of independence. Disponível em: http://www.r-tutor.com/elementary-statistics/goodness-fit/chi-squared-test-independence. Acesso em Novembro de 2014.

R (2014b). An r introduction to statistics. Disponível em: http://www.r-tutor.com/. Acesso em Janeiro de 2014.

Riaz, M.; Mendes, E. & Tempero, E. D. (2009). A systematic review of software maintainability prediction and metrics. Em *ESEM*, pp. 367–377.

Riehle, D. (2000). Framework design: A role modeling approach. *Softwaretechnik-Trends*, 20(4).

Rosenberg, L.; Ruth, S. & Gallo, A. (1999). Risk-based Object Oriented Testing. Em *In: Proceedings of the 24 th annual Software Engineering Workshop, NASA, Software Engineering Laboratory*.

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.

Sommerville, I. (2010). *Software Engineering*. Addison-Wesley, Harlow, England, 9 edição. ISBN 978-0-13-703515-1.

Spinellis, D. (2008). A tale of four kernels. Em *Proceedings of the 30th international conference on Software engineering*, ICSE '08, pp. 381--390, New York, NY, USA. ACM.

Tempero, E.; Anslow, C.; Dietrich, J.; Han, T.; Li, J.; Lumpe, M.; Melton, H. & Noble, J. (2010). Qualitas corpus: A curated collection of java code for empirical studies. Em *2010 Asia Pacific Software Engineering Conference (APSEC2010)*, pp. 336--345.

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

Van, M. & Vose, D. (2008). *A Compendium of Distributions*. John Wiley and Sons.

VoseSoftware (2014). Vosesoftware. Disponível em: http://www.vosesoftware.com/ModelRiskHelp. Acesso em Janeiro de 2014.

Apêndice A

Processo de Identificação dos Valores Referência

A.1 Valores Referência

Esse apêndice apresenta o processo de identificação dos valores referência para as métricas de softwares orientados por objetos que não foram detalhadas no Capítulo 5.

A.1.1 Acoplamento Eferente (CE)

O Gráfico de Dispersão dos valores coletados para a métrica Acoplamento Eferente (CE), ilustrado na Figura A.1a, sugere uma distribuição de cauda-pesada. A Figura A.1b ressalta ainda mais essa característica, pois é possível observar que a aproximação dos 100% de frequência acumulada ao longo do eixo x (valores das métricas) ocorre de uma forma drasticamente rápida, ou seja, há uma aproximação quase que instantânea dos 100% dos valores. Isso significa que há um grande número de pacotes acoplados a poucas classes "externas" a ele e um número pequeno de pacotes acoplados a muitas classes "externas" a ele.

Como mostrado nas Figuras A.1c e A.1d, o conjunto de valores dessa métrica possui melhor ajuste à distribuição *Generalized Extreme Value*, com parâmetros $k = 0,527$, $\sigma = 2,834$ e $\mu = 2,397$, conforme retornado pela ferramenta *EasyFit*. Essa distribuição tem característica cauda-pesada. A Figura A.2 mostra a Função Densidade de Probabilidade com os parâmetros retornados, onde é possível visualizar a característica de cauda-pesada da distribuição com esses parâmetros.

A Figura A.1e mostra o histograma de probabilidade dos dados. Percebe-se que a segunda barra, que representa os valores de CE entre 5 e 10, constitui 20% dos dados,

enquanto a terceira barra, que representa os valores entre 10 e 15, constitui menos de 10% dos dados. Novamente, fica ressaltada a característica da distribuição ser de cauda-pesada. A Figura A.1f mostra esses dados em escala logarítmica. Tal como na métrica CA, percebe-se uma reta inclinada à esquerda, característica de uma lei de potência.

Foram identificados os valores que representam 70° e 90° percentil dos dados, que correspondem aos valores 6 e 16. Isso significa que 70% dos pacotes possuem $CE \leq 6$ e 90% dos pacotes possuem $CE \leq 16$. Os valores 70% e 90% foram identificados por meio de uma análise visual no Gráfico de Frequência Relativa Acumulada da Figura A.1b. Os pontos marcados com a cor verde / quadrado e vermelha / losango representam as regiões identificadas nessa análise. A Tabela A.1 sumariza os valores referência derivados para a métrica Acoplamento Eferente (CE).

Tabela A.1: Valores referência para CE

Bom/Frequente	**Regular/Ocasional**	**Ruim/Raro**
$CE \leq 6$	$6 < CE \leq 16$	$CE > 16$

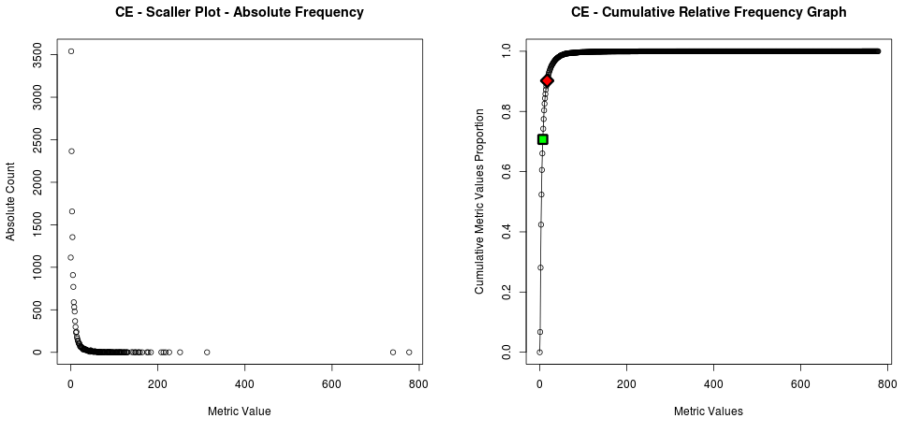
.

A.1.2 Linhas de Código por Método (MLOC)

O Gráfico de Dispersão dos valores coletados para a métrica Linhas de Código por Método (MLOC), na Figura A.3a, sugere fortemente uma distribuição de cauda-pesada. A Figura A.3b ressalta ainda mais essa característica, pois permite observar que a aproximação dos 100% de frequência acumulada ao longo do eixo x (valores das métricas) ocorre de uma forma drasticamente rápida. Isso significa que há um grande número de métodos com poucas linhas de código e um pequeno número de métodos com muitas linhas de código.

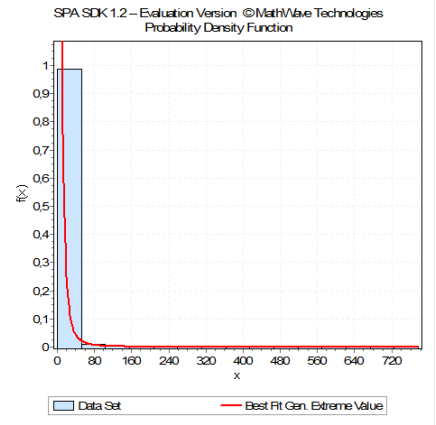
Como mostrado nas Figuras A.3c e A.3d, o conjunto de valores dessa métrica possui melhor ajuste à distribuição *Pareto 2*, com parâmetros $\alpha = 1,226$ e $\beta = 3,051$, conforme retornado pela ferramenta *EasyFit*. Essa distribuição tem característica de cauda-pesada. A Figura A.4 mostra a Função Densidade de Probabilidade com os parâmetros retornados, sendo possível visualizar a característica de cauda-pesada da distribuição com esses parâmetros.

A Figura A.3e mostra o histograma de probabilidade dos dados. Percebe-se que a segunda barra, que representa os valores de MLOC entre 5 e 10, constitui menos de 15% dos dados, enquanto a terceira barra, que representa os valores entre 10 e 15, constitui menos de 10% dos dados. Novamente, fica ressaltada a característica da distribuição

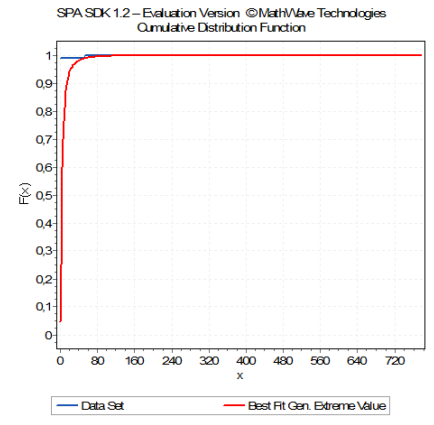


(a)

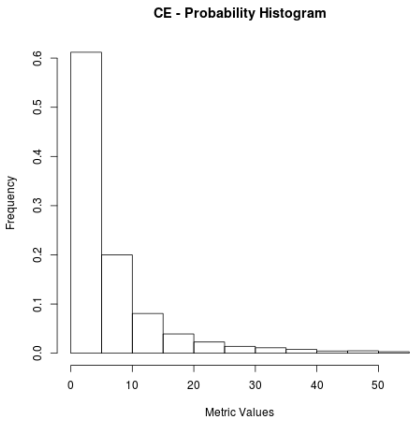
(b)



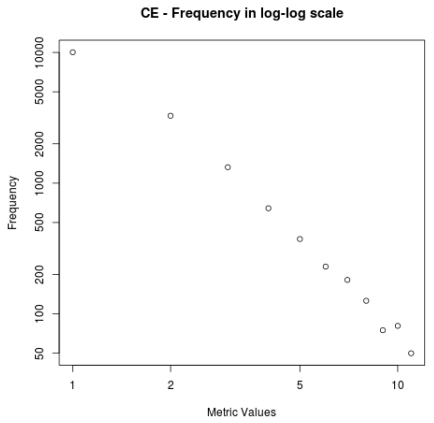
(c)



(d)



(e)



(f)

Figura A.1: Acoplamento Eferente: (a) Gráfico de Dispersão (b) Gráfico de Frequência Relativa Acumulada (c) Função de Densidade de Probabilidade ajustada à Distribuição Generalized Extreme Value (d) Gráfico de Frequência Relativa Acumulada ajustado à Distribuição Generalized Extreme Value (e) Histograma de Probabilidade (f) Histograma em escala *log-log*.

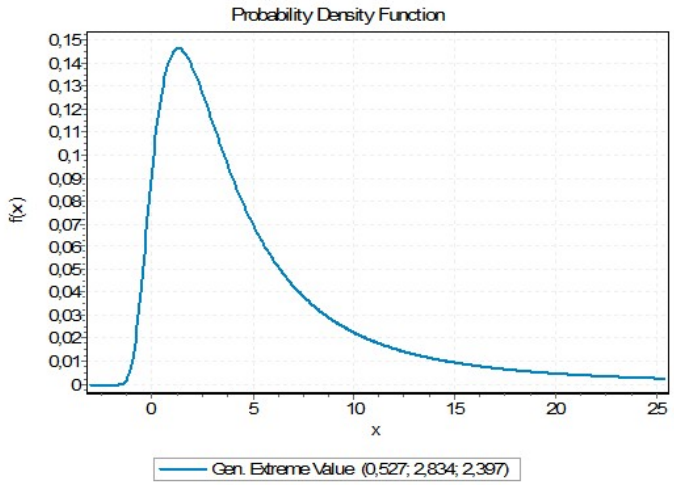


Figura A.2: CE - Função de Densidade de Probabilidade com os parâmetros retornados pela ferramenta *EasyFit*

ser cauda-pesada. A Figura A.3f mostra esses dados em escala logarítmica. Percebe-se uma reta inclinada à esquerda quase que perfeita, característica de uma lei de potência.

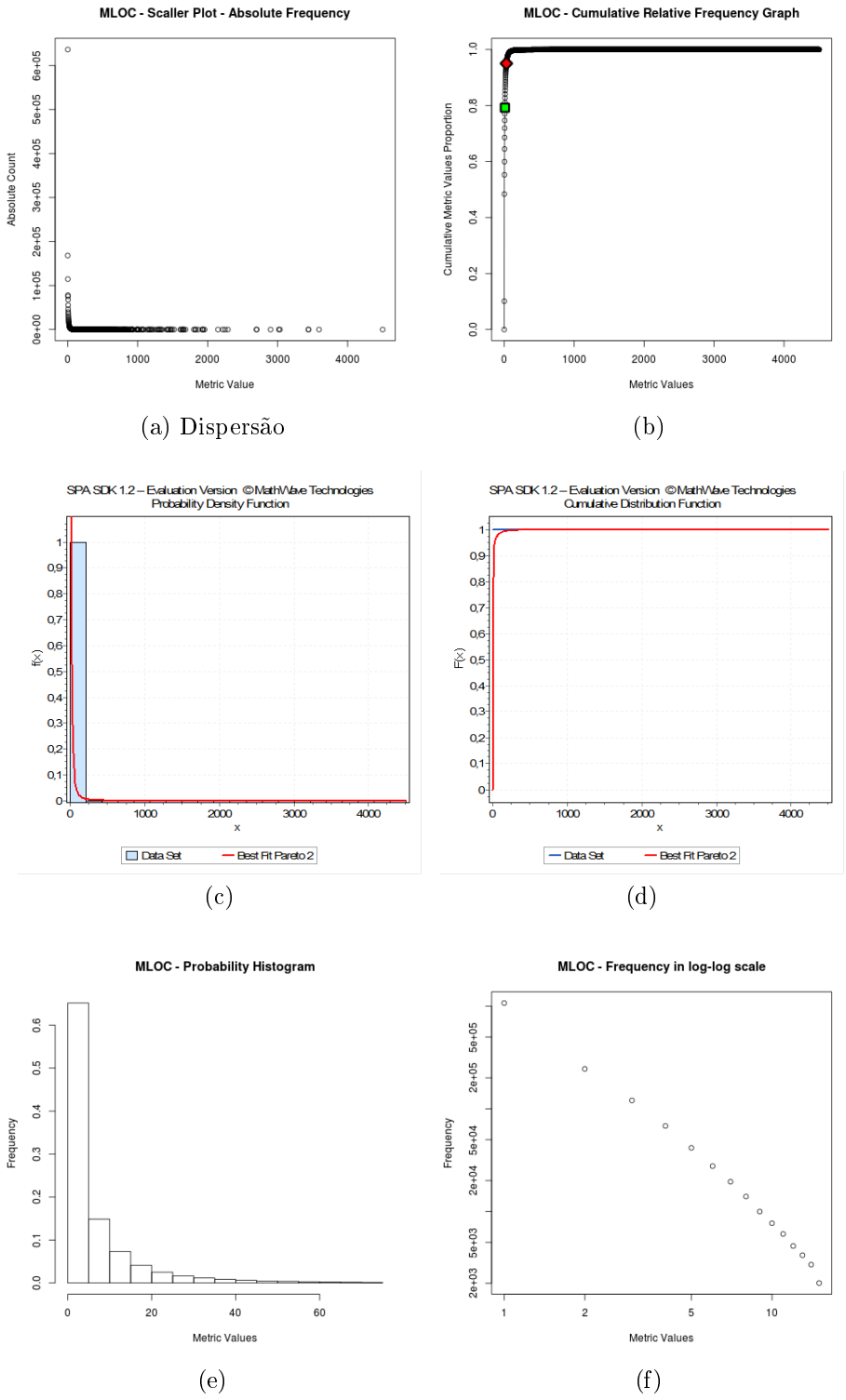
Foram identificados os valores que representam o 80° e 95° percentil dos dados, correspondendo aos valores 10 e 30. Nessa métrica, foram usados percentis diferentes das métricas anteriores pois o Gráfico da Figura A.3b tem uma aproximação quase que instantânea dos 100% dos dados para valores baixos. Se fossem utilizados o 70° e 90° percentis, tal como nas métricas CA e CE, os valores resultantes seriam 6 e 19, que são baixos. Por isso, e pela rápida aproximação observada no gráfico, é mais apropriado utilizar o 80° e 95° percentil. Os pontos marcados com a cor verde / quadrado e vermelha / losango representam as regiões identificadas nessa análise visual da rápida aproximação no Gráfico de Frequência Relativa Acumulada da Figura A.3b. Isso significa que 80% dos métodos possuem $MLOC \leq 10$ e 95% dos métodos possuem $MLOC \leq 30$. A Tabela A.2 sumariza os valores referência derivados para a métrica Linhas de Código por Método (MLOC).

Tabela A.2: Valores referência para MLOC

Bom/Frequente	**Regular/Ocasional**	**Ruim/Raro**
$MLOC \leq 10$	$10 < MLOC \leq 30$	$MLOC > 30$

A.1.3 Número de Classes por Pacote (NOC)

O Gráfico de Dispersão dos valores coletados para a métrica Número de Classes por Pacote (NOC), ilustrado na Figura A.5a, sugere uma distribuição de cauda-pesada.



(a) Dispersão (b)

(c) (d)

(e) (f)

Figura A.3: Linhas de Código por Método: (a) Gráfico de Dispersão (b) Gráfico de Frequência Relativa Acumulada (c) Função de Densidade de Probabilidade ajustada à Distribuição Pareto 2 (d) Gráfico de Frequência Relativa Acumulada ajustado à Distribuição Pareto 2 (e) Histograma de Probabilidade (f) Histograma em escala *log-log*.

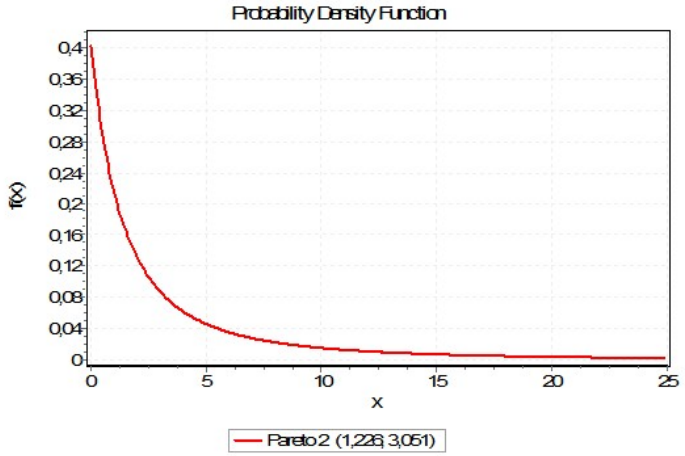


Figura A.4: MLOC - Função de Densidade de Probabilidade com os parâmetros retornados pela ferramenta *EasyFit*

A Figura A.5b ressalta ainda mais essa característica, pois é possível observar que a aproximação dos 100% de frequência acumulada ao longo do eixo x (valores das métricas) ocorre rapidamente. Isso significa que há um grande número de pacotes com uma quantidade pequena de classes e um pequeno número de pacotes com muitas classes.

Como mostrado nas Figuras A.5c e A.5d, o conjunto de valores dessa métrica possui melhor ajuste à distribuição *Log-Logistic*, com parâmetros $\alpha = 1,452$ e $\beta = 5,520$, conforme retornado pela ferramenta *EasyFit*. Essa distribuição tem característica de cauda-pesada, o que está de acordo com as observações dos Gráficos de Dispersão e Frequência Relativa Acumulada. A Figura A.6 mostra a Função Densidade de Probabilidade com os parâmetros retornados, onde é possível visualizar a característica de cauda-pesada da distribuição com esses parâmetros.

A Figura A.5e mostra o histograma de probabilidade dos dados. Percebe-se que a segunda barra, que representa os valores de NOC entre 5 e 10, constitui pouco mais de 20% dos dados, enquanto a terceira barra, que representa os valores entre 10 e 15, constitui pouco mais de 10% dos dados. Novamente, fica ressaltada a característica da distribuição ser de cauda-pesada. A Figura A.5f mostra esses dados em escala logarítmica. Percebe-se uma reta inclinada à esquerda quase que perfeita, característica de uma lei de potência.

Foram identificados os valores que representam o 70° e 90° percentil dos dados, que correspondem aos valores 11 e 28. Isso significa que 70% dos pacotes possuem $NOC \leq 11$ e 90% dos métodos possuem $NOC \leq 28$. Os valores 70% e 90% foram identificados por meio de uma análise visual no Gráfico de Frequência Relativa Acu-

mulada da Figura A.5b. Os pontos marcados com a cor verde / quadrado e vermelha / losango representam as regiões identificadas nessa análise. A Tabela A.3 sumariza os valores referência derivados para a métrica Número de Classes por Pacote (NOC).

Tabela A.3: Valores referência para NOC

Bom/Frequente	**Regular/Ocasional**	**Ruim/Raro**
$NOC \leq 11$	$11 < NOC \leq 28$	$NOC > 28$

A.1.4 Número de Atributos (NOF)

O Gráfico de Dispersão dos valores coletados para a métrica Número de Atributos (NOF), ilustrado na Figura A.7a, bem como o Gráfico de Frequência Relativa Acumulada, na Figura A.7b, sugerem uma distribuição cauda-pesada. Isso significa que há um grande número de classes com uma quantidade reduzida de atributos e um pequeno número de classes com um grande quantidade de atributos.

Como mostrado nas Figuras A.7c e A.7d, o conjunto de valores dessa métrica possui melhor ajuste à distribuição *Beta*, com parâmetros $\alpha_1 = 0,059$, $\alpha_2 = 64,580$, $a = 0,000$ e $b = 2026,446$, conforme retornado pela ferramenta *EasyFit*. Pela linha do ajuste dos dados à distribuição, é possível observar a característica de cauda-pesada. A Figura A.8 mostra a Função Densidade de Probabilidade com os parâmetros retornados, onde também é possível visualizar a característica de cauda-pesada da distribuição com esses parâmetros.

A Figura A.7e mostra o histograma de probabilidade dos dados. Percebe-se que a segunda barra, que representa o valor 1, constitui pouco mais de 10% dos dados, enquanto a terceira barra, que representa o valor 2, constitui pouco menos de 10% dos dados, formando a cauda na distribuição dos dados. A Figura A.7f mostra esses dados em escala logarítmica, sugerindo uma lei de potência.

Foram identificados os valores que representam o 80° e 95° percentil dos dados, que correspondem aos valores 3 e 8. Isso significa que 80% das classes possuem $NOF \leq 3$ e 95% das classes possuem $NOF \leq 8$. Os valores 80% e 95% foram identificados por meio de uma análise visual no Gráfico de Frequência Relativa Acumulada da Figura A.7b. Os pontos marcados com a cor verde / quadrado e vermelha / losango representam as regiões identificadas nessa análise. A Tabela A.4 sumariza os valores referência derivados para a métrica Número de Atributos (NOF).

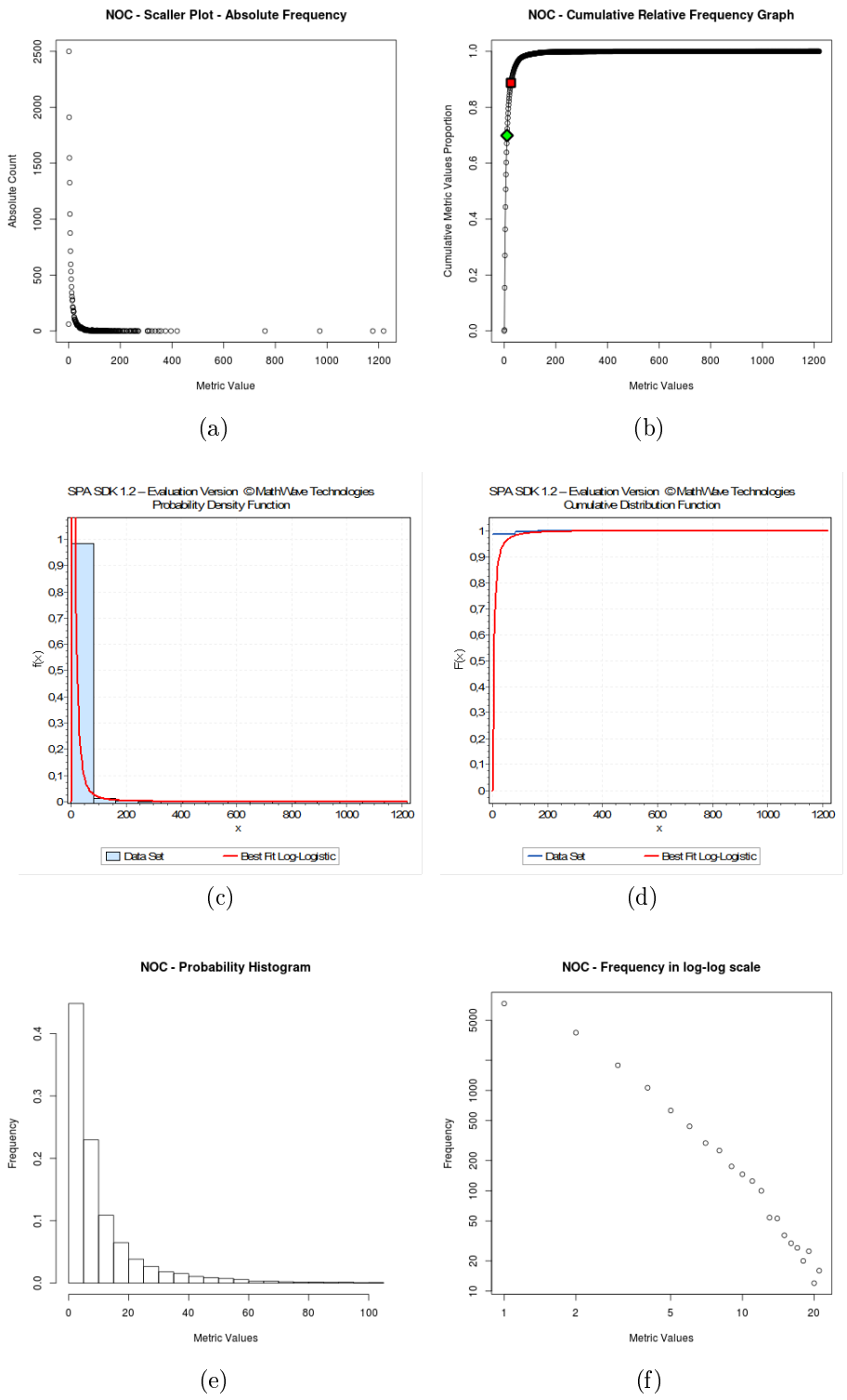


Figura A.5: Número de Classes por Pacote: (a) Gráfico de Dispersão (b) Gráfico de Frequência Relativa Acumulada (c) Função de Densidade de Probabilidade ajustada à Distribuição Log-Logistic (d) Gráfico de Frequência Relativa Acumulada ajustado à Distribuição Log-Logistic (e) Histograma de Probabilidade (f) Histograma em escala *log-log*.

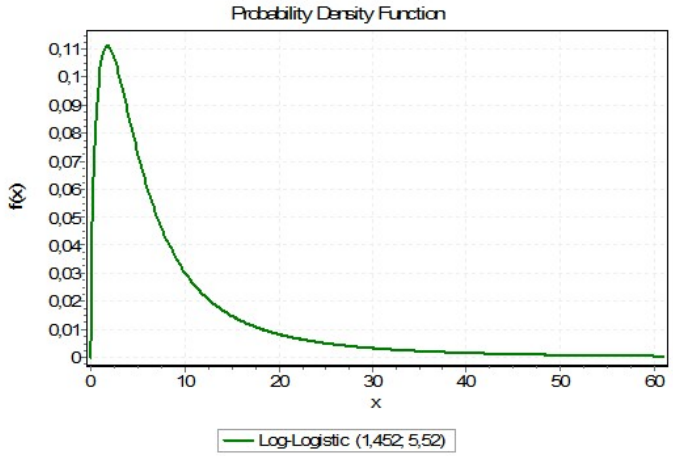


Figura A.6: NOC - Função de Densidade de Probabilidade com os parâmetros retornados pela ferramenta *EasyFit*

Tabela A.4: Valores referência para NOF

Bom/Frequente	**Regular/Ocasional**	**Ruim/Raro**
$NOF \leq 3$	$3 < NOF \leq 8$	$NOF > 8$

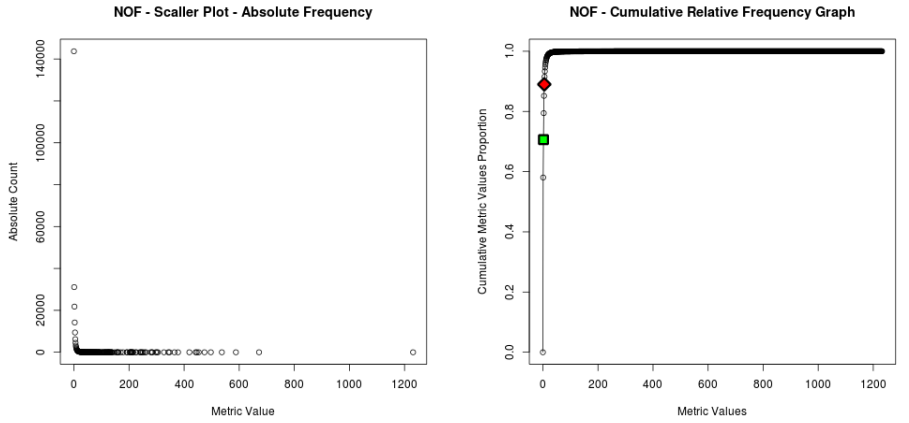
A.1.5 Número de Métodos (NOM)

O Gráfico de Dispersão dos valores coletados para a métrica Número de Métodos (NOM), ilustrado na Figura A.9a, bem como o Gráfico de Frequência Relativa Acumulada, Figura A.9b, sugerem uma distribuição de cauda-pesada. Isso significa que há um grande número de classes com poucos métodos e um pequeno número de classes com muitos métodos.

Como mostrado nas Figuras A.9c e A.9d, o conjunto de valores dessa métrica possui melhor ajuste à distribuição *Weibull*, com parâmetros $\alpha = 0,852$ e $\beta = 5,879$, conforme retornado pela ferramenta *EasyFit*. Pela linha do ajuste dos dados à distribuição, é possível identificar a característica de cauda-pesada. A Figura A.10 mostra a Função Densidade de Probabilidade com os parâmetros retornados, onde é possível visualizar a característica de cauda-pesada da distribuição com esses parâmetros.

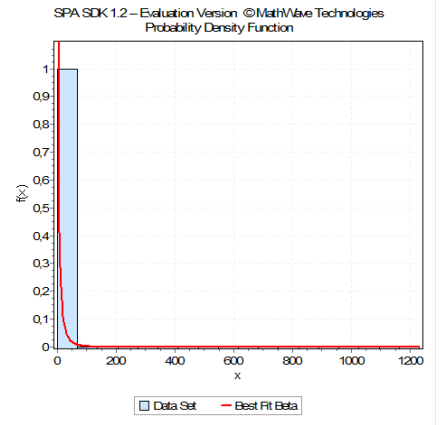
A Figura A.9e mostra o histograma de probabilidade dos dados. Percebe-se que a segunda barra, que representa os valores entre 2 e 4, constitui pouco menos de 25% dos dados, enquanto a terceira barra, que representa os valores entre 4 e 6, constitui pouco menos de 15% dos dados, formando a cauda na distribuição dos dados. A Figura A.9f mostra esses dados em escala logarítmica, sugerindo uma lei de potência.

Foram identificados os valores que representam o 70° e 90° percentil dos dados, que correspondem aos valores 6 e 14. Isso significa que 70% das classes possuem

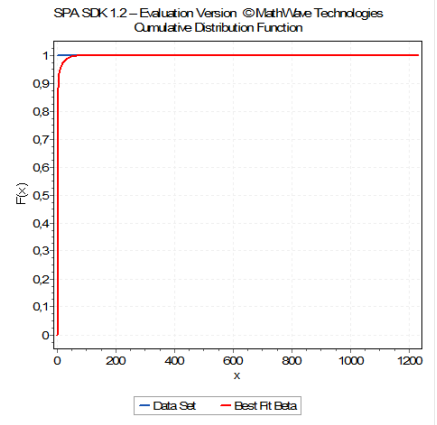


(a)

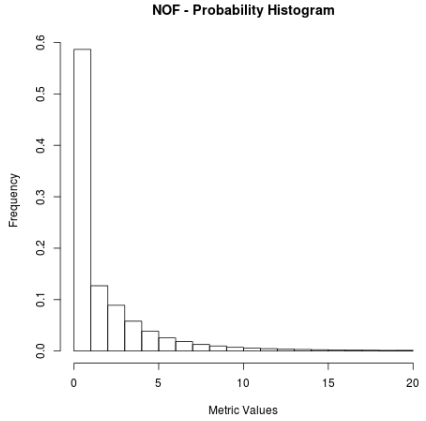
(b)



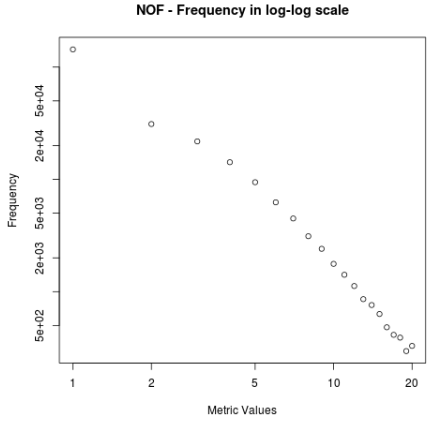
(c)



(d)



(e)



(f)

Figura A.7: Número de Atributos: (a) Gráfico de Dispersão (b) Gráfico de Frequência Relativa Acumulada (c) Função de Densidade de Probabilidade ajustada à Distribuição Beta (d) Gráfico de Frequência Relativa Acumulada ajustado à Distribuição Beta (e) Histograma de Probabilidade (f) Histograma em escala *log-log*.

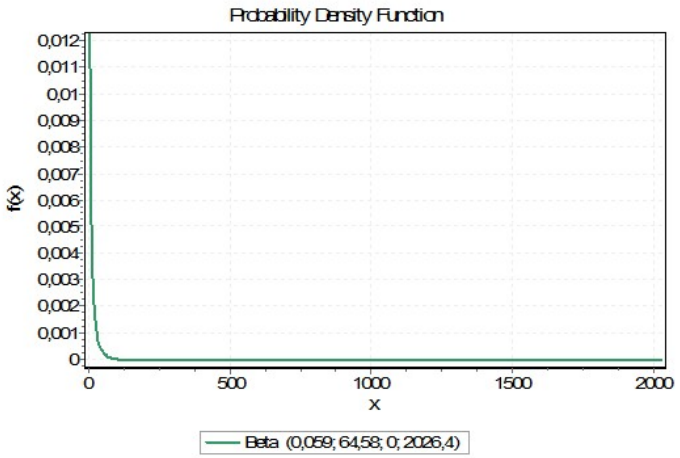


Figura A.8: NOF - Função de Densidade de Probabilidade com os parâmetros retornados pela ferramenta *EasyFit*

$NOM \leq 6$ e 90% das classes possuem $NOM \leq 14$. Os valores 70% e 90% foram identificados por meio de uma análise visual no Gráfico de Frequência Relativa Acumulada da Figura A.9b. Os pontos marcados com a cor verde / quadrado e vermelha / losango representam as regiões identificadas nessa análise. A Tabela A.5 sumariza os valores referência derivados para a métrica Número de Métodos (NOM).

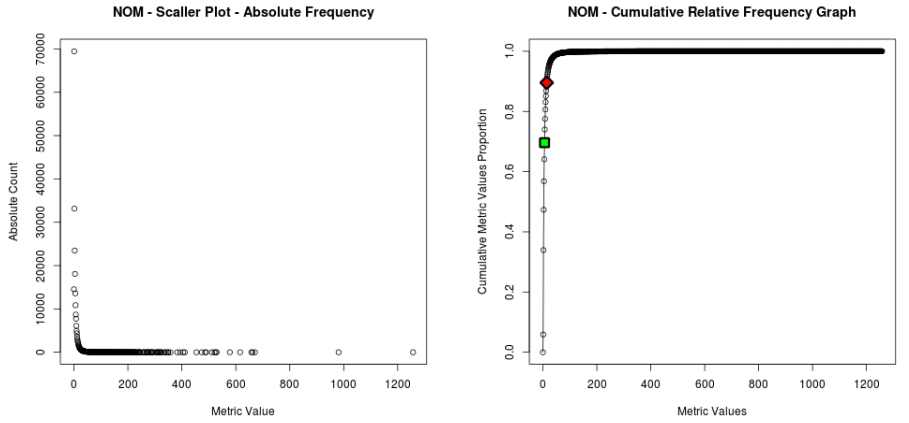
Tabela A.5: Valores referência para NOM

Bom/Frequente	**Regular/Ocasional**	**Ruim/Raro**
$NOM \leq 6$	$6 < NOM \leq 14$	$NOM > 14$

A.1.6 Número de Métodos Sobrescritos (NORM)

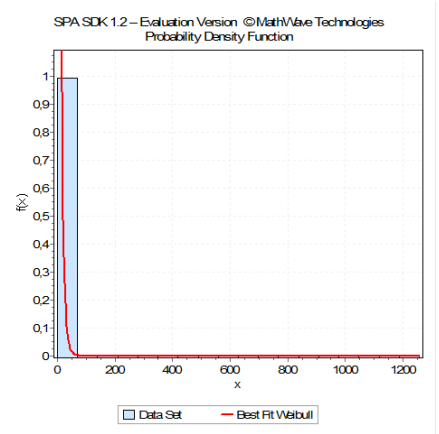
O Gráfico de Dispersão dos valores coletados para a métrica Número de Métodos Sobrescritos (NORM), ilustrado na Figura A.11a, bem como o Gráfico de Frequência Relativa Acumulada, Figura A.11b, sugerem uma distribuição de cauda-pesada. Isso significa que há um grande número de classes com poucos métodos sobrescritos e um pequeno número de classes com muitos métodos sobrescritos.

Como mostrado nas Figuras A.11c e A.11d, o conjunto de valores dessa métrica possui melhor ajuste à distribuição *Power Function*, com parâmetros $\alpha = 0,006$, $a = 0,000$ e $b = 235,200$, conforme retornado pela ferramenta *EasyFit*. Pela linha do ajuste dos dados à distribuição, é possível observar a característica de cauda-pesada. A Figura A.12 mostra a Função Densidade de Probabilidade com os parâmetros retornados, onde é possível visualizar a característica de cauda-pesada da distribuição com esses parâmetros.

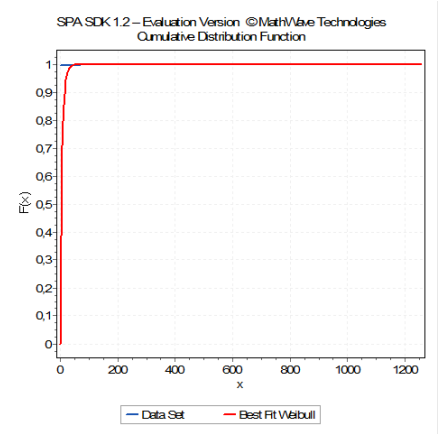


(a)

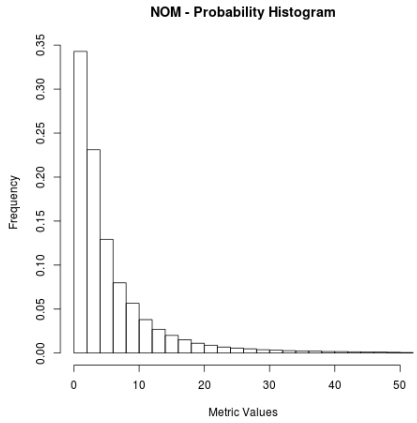
(b)



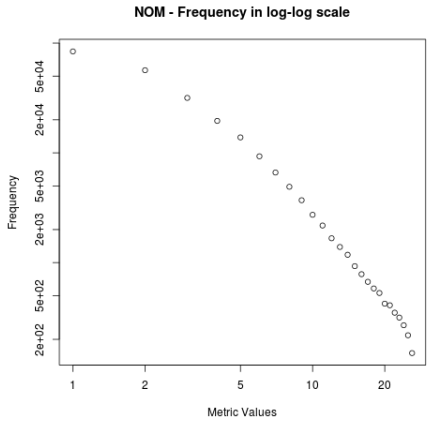
(c)



(d)



(e)



(f)

Figura A.9: Número de Métodos: (a) Gráfico de Dispersão (b) Gráfico de Frequência Relativa Acumulada (c) Função de Densidade de Probabilidade ajustada à Distribuição Weibull (d) Gráfico de Frequência Relativa Acumulada ajustado à Distribuição Weibull (e) Histograma de Probabilidade (f) Histograma em escala *log-log*.

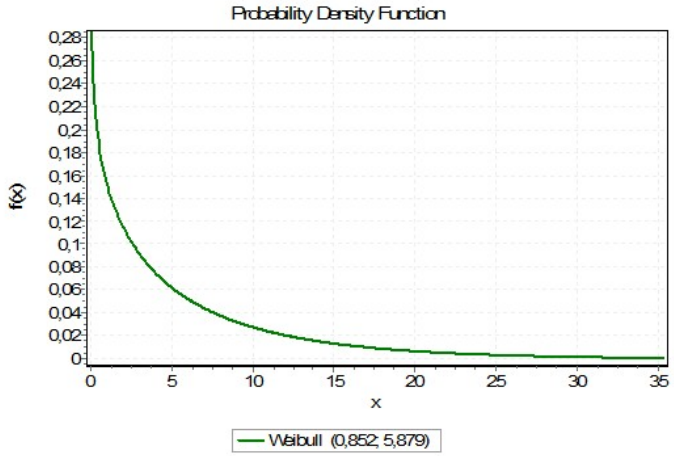


Figura A.10: NOM - Função de Densidade de Probabilidade com os parâmetros retornados pela ferramenta *EasyFit*

A Figura A.11e mostra o histograma de probabilidade dos dados. Percebe-se que a segunda barra, que representa os valores 1, constitui pouco menos de 20% dos dados, enquanto a terceira barra, que representa os valores 2, constitui pouco menos de 10% dos dados, formando a cauda na distribuição dos dados. A Figura A.11f mostra esses dados em escala logarítmica, sugerindo uma lei de potência.

Foram identificados os valores que representam o 90° e 97° percentil dos dados, que correspondem aos valores 2 e 4. Isso significa que 90% das classes possuem $NORM \leq 2$ e 97% das classes possuem $NORM \leq 4$. Os valores 90% e 97% foram identificados por meio de uma análise visual no Gráfico de Frequência Relativa Acumulada da Figura A.11b. Os pontos marcados com a cor verde / quadrado e vermelha / losango representam as regiões identificadas nessa análise. A Tabela A.6 sumariza os valores referência derivados para a métrica Número de Métodos Sobrescritos (NORM).

Tabela A.6: Valores referência para NORM

Bom/Frequente	**Regular/Ocasional**	**Ruim/Raro**
$NORM \leq 2$	$2 < NORM \leq 4$	$NORM > 4$

A.1.7 Número de Filhas (NSC)

O Gráfico de Dispersão dos valores coletados para a métrica Número de Filhas (NSC), na Figura A.13a, bem como o Gráfico de Frequência Relativa Acumulada, na Figura A.13b, sugerem uma distribuição cauda-pesada. Isso significa que há um grande número de classes com poucas classes filhas e um número pequeno de classes com muitas classes filhas.

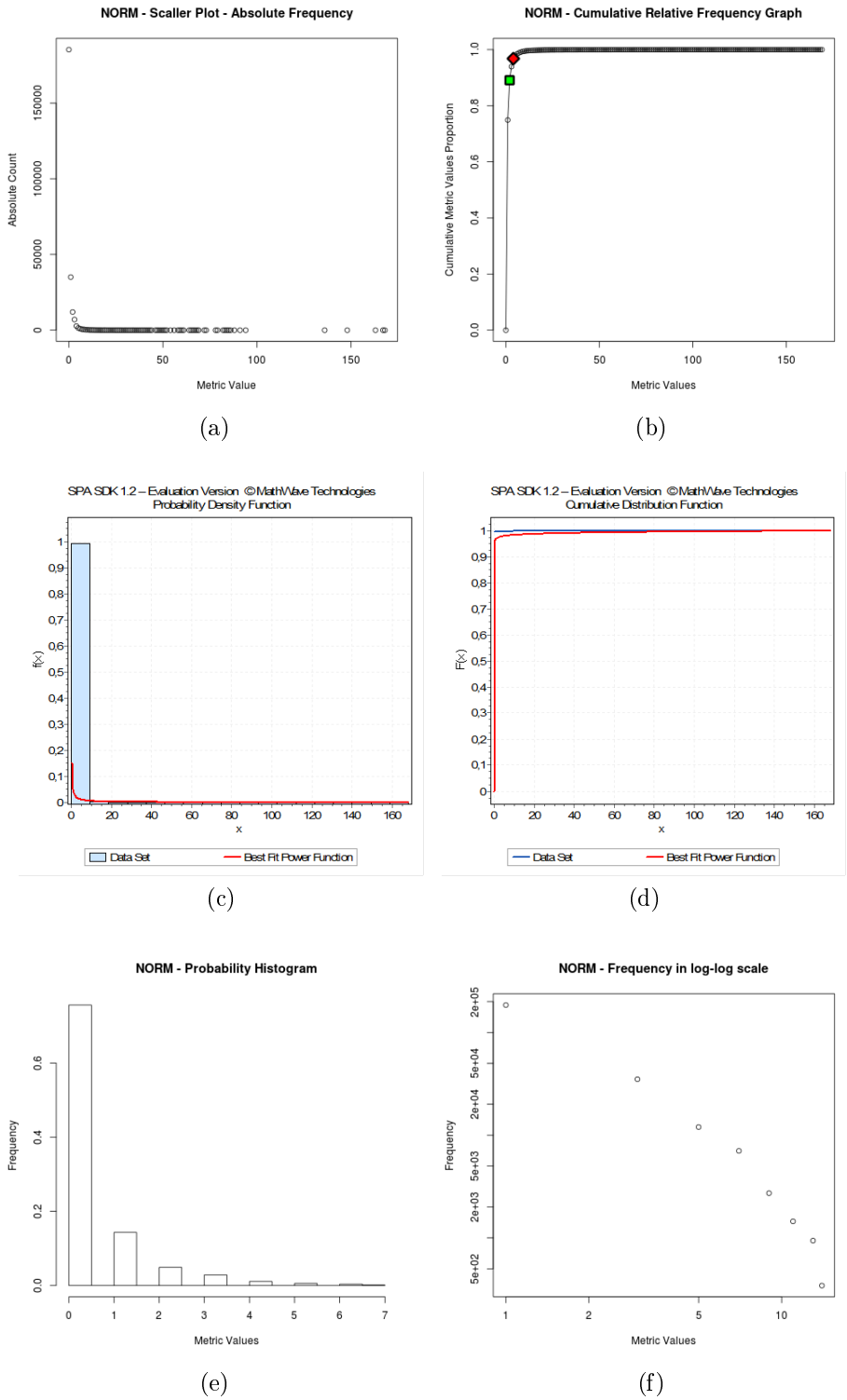


Figura A.11: Número de Métodos Sobrescritos: (a) Gráfico de Dispersão (b) Gráfico de Frequência Relativa Acumulada (c) Função de Densidade de Probabilidade ajustada à Distribuição Power Function (d) Gráfico de Frequência Relativa Acumulada ajustado à Distribuição Power Function (e) Histograma de Probabilidade (f) Histograma em escala *log-log*.

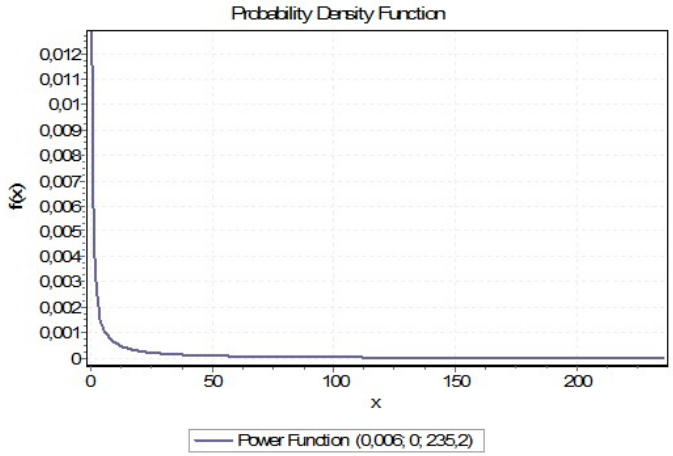


Figura A.12: NORM - Função de Densidade de Probabilidade com os parâmetros retornados pela ferramenta *EasyFit*

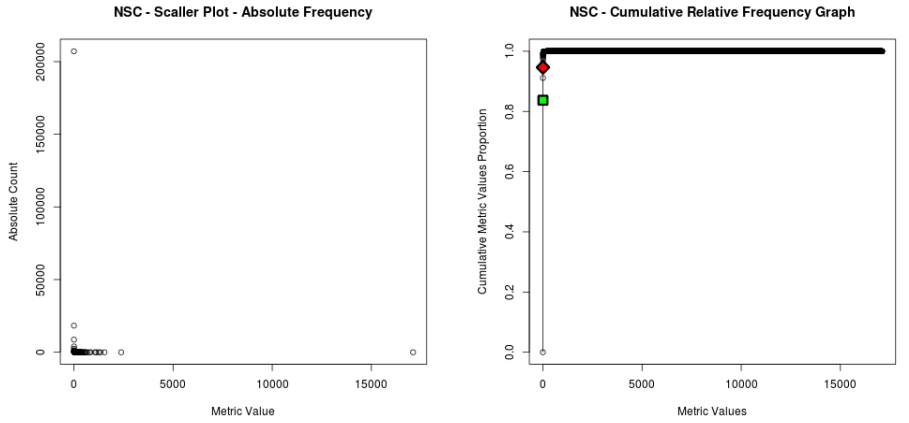
Como mostrado nas Figuras A.13c e A.13d, o conjunto de valores dessa métrica possui melhor ajuste à distribuição *Beta*, com parâmetros $\alpha_1 = 0,001$, $\alpha_2 = 9,467$, $a = 0,000$ e $b = 25465,529$, conforme retornado pela ferramenta *EasyFit*. Pela linha do ajuste dos dados à distribuição, é possível observar a característica de cauda-pesada. A Figura A.14 mostra a Função Densidade de Probabilidade com os parâmetros retornados, onde é possível visualizar a característica de cauda-pesada da distribuição com esses parâmetros.

A Figura A.13e mostra o histograma de probabilidade dos dados. Percebe-se que a segunda barra, que representa os valores 1, constitui pouco menos de 10% dos dados, enquanto a terceira barra, representando os valores 2, constitui pouco menos de 5% dos dados, formando a cauda na distribuição dos dados. A Figura A.13f mostra esses dados em escala logarítmica, sugerindo uma lei de potência.

Foram identificados os valores que representam o 85° e 95° percentil dos dados, que correspondem aos valores 1 e 3. Isso significa que 85% das classes possuem $NSC \leq 1$ e 95% das classes possuem $NSC \leq 3$. Os valores 85% e 95% foram identificados por meio de uma análise visual no Gráfico de Frequência Relativa Acumulada da Figura A.13b. Os pontos marcados com a cor verde / quadrado e vermelha / losango representam as regiões identificadas nessa análise. A Tabela A.7 sumariza os valores referência derivados para a métrica Número de Filhas (NSC).

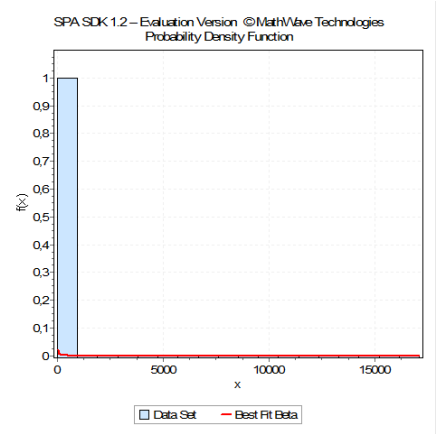
Tabela A.7: Valores referência para NSC

Bom/Frequente	**Regular/Ocasional**	**Ruim/Raro**
$NSC \leq 1$	$1 < NSC \leq 3$	$NSC > 3$

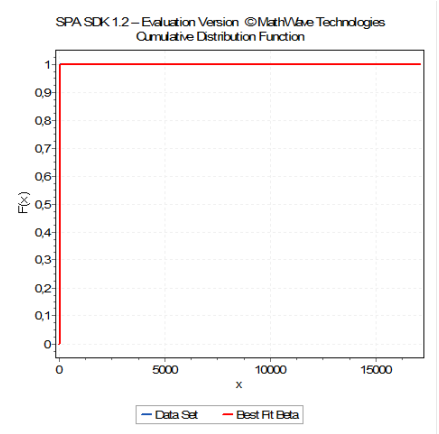


(a)

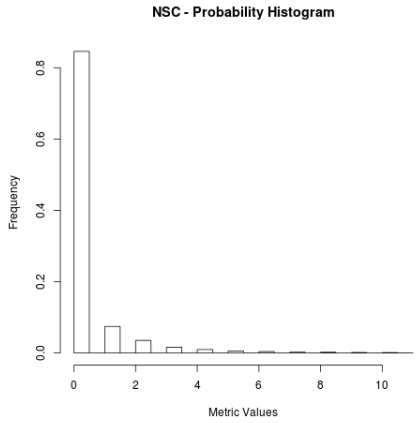
(b)



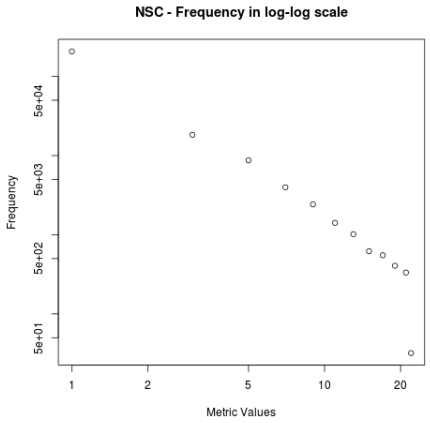
(c)



(d)



(e)



(f)

Figura A.13: Número de Filhas: (a) Gráfico de Dispersão (b) Gráfico de Frequência Relativa Acumulada (c) Função de Densidade de Probabilidade ajustada à Distribuição Beta (d) Gráfico de Frequência Relativa Acumulada ajustado à Distribuição Beta (e) Histograma de Probabilidade (f) Histograma em escala *log-log*.

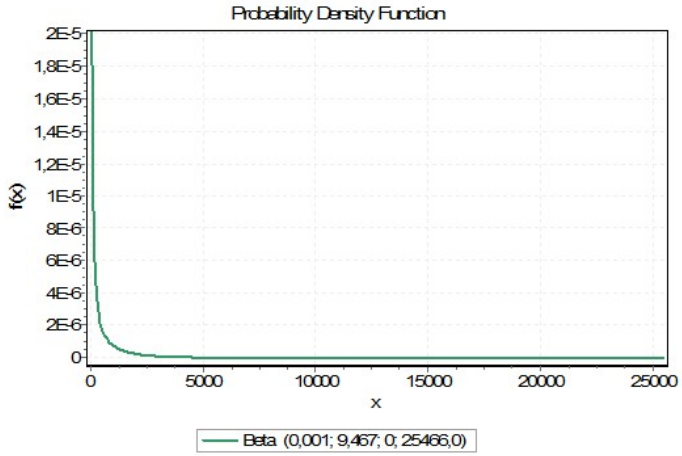


Figura A.14: NSC - Função de Densidade de Probabilidade com os parâmetros retornados pela ferramenta *EasyFit*

A.1.8 Número de Atributos Estáticos (NSF)

O Gráfico de Dispersão dos valores coletados para a métrica Número de Atributos Estáticos (NSF), ilustrado na Figura A.15a, bem como o Gráfico de Frequência Relativa Acumulada, Figura A.15b, sugerem uma distribuição cauda-pesada. Isso significa que há um grande número de classes com poucos atributos estáticos e um número pequeno de classes com muitos atributos estáticos.

Como mostrado nas Figuras A.15c e A.15d, o conjunto de valores dessa métrica possui melhor ajuste à distribuição *Beta*, com parâmetros $\alpha_1 = 0,018$, $\alpha_2 = 18,284$, $a = 0,000$ e $b = 4130,615$, conforme retornado por *EasyFit*. Pela linha do ajuste dos dados à distribuição, identifica-se a característica de cauda-pesada. A Figura A.16 mostra a Função Densidade de Probabilidade com os parâmetros retornados, onde é possível visualizar a característica de cauda-pesada da distribuição com esses parâmetros.

A Figura A.15e mostra o histograma de probabilidade dos dados. Percebe-se que a segunda barra, que representa os valores 1, constitui pouco mais de 10% dos dados, enquanto a terceira barra, representando os valores 2, constitui pouco menos de 5% dos dados, formando a cauda na distribuição dos dados. A Figura A.15f mostra esses dados em escala logarítmica, sugerindo uma lei de potência.

Foram identificados os valores que representam o 85° e 95° percentil dos dados, que correspondem aos valores 1 e 5. Isso significa que 85% das classes possuem $NSF \leq 1$ e 95% das classes possuem $NSF \leq 5$. Os valores 85% e 95% foram identificados por meio de uma análise visual no Gráfico de Frequência Relativa Acumulada da Figura A.15b. Os pontos marcados com a cor verde / quadrado e vermelha / losango representam as regiões identificadas nessa análise. A Tabela A.8 sumariza os valores

referência derivados para a métrica Número de Atributos Estáticos (NSF).

Tabela A.8: Valores referência para NSF

Bom/Frequente	**Regular/Ocasional**	**Ruim/Raro**
$NSF \leq 1$	$1 < NSF \leq 5$	$NSF > 5$

A.1.9 Número de Métodos Estáticos (NSM)

O Gráfico de Dispersão dos valores coletados para a métrica Número de Métodos Estáticos (NSM), observado na Figura A.17a, bem como o Gráfico de Frequência Relativa Acumulada, Figura A.17b, sugerem uma distribuição cauda-pesada. Isso significa que há um grande número de classes com poucos métodos estáticos e um número pequeno de classes com muitos métodos estáticos.

Como mostrado nas Figuras A.17c e A.17d, o conjunto de valores dessa métrica possui melhor ajuste à distribuição *Beta*, com parâmetros $\alpha_1 = 0,017$, $\alpha_2 = 27,961$, $a = 0,000$ e $b = 1646,287$, conforme retornado por *EasyFit*. Pela linha do ajuste dos dados à distribuição, é possível observar a característica de cauda-pesada. A Figura A.18 mostra a Função Densidade de Probabilidade com os parâmetros retornados, onde é possível visualizar a característica de cauda-pesada da distribuição com esses parâmetros.

A Figura A.17e mostra o histograma de probabilidade dos dados. Percebe-se que a segunda barra, que representa os valores 1, constitui menos de 10% dos dados, enquanto a terceira barra, que representa os valores 2, constitui menos de 5% dos dados, formando a cauda na distribuição dos dados. A Figura A.17f mostra esses dados em escala logarítmica, sugerindo uma lei de potência.

Foram identificados os valores que representam o 90° e 96° percentil dos dados, que correspondem aos valores 1 e 3. Isso significa que 90% das classes possuem $NSM \leq 1$ e 96% das classes possuem $NSM \leq 3$. Os valores 90% e 96% foram identificados por meio de uma análise visual no Gráfico de Frequência Relativa Acumulada da Figura A.17b. Os pontos marcados com a cor verde / quadrado e vermelha / losango representam as regiões identificadas nessa análise. A Tabela A.9 sumariza os valores referência derivados para a métrica Número de Métodos Estáticos (NSM).

Tabela A.9: Valores referência para NSM

Bom/Frequente	**Regular/Ocasional**	**Ruim/Raro**
$NSM \leq 1$	$1 < NSM \leq 3$	$NSM > 3$

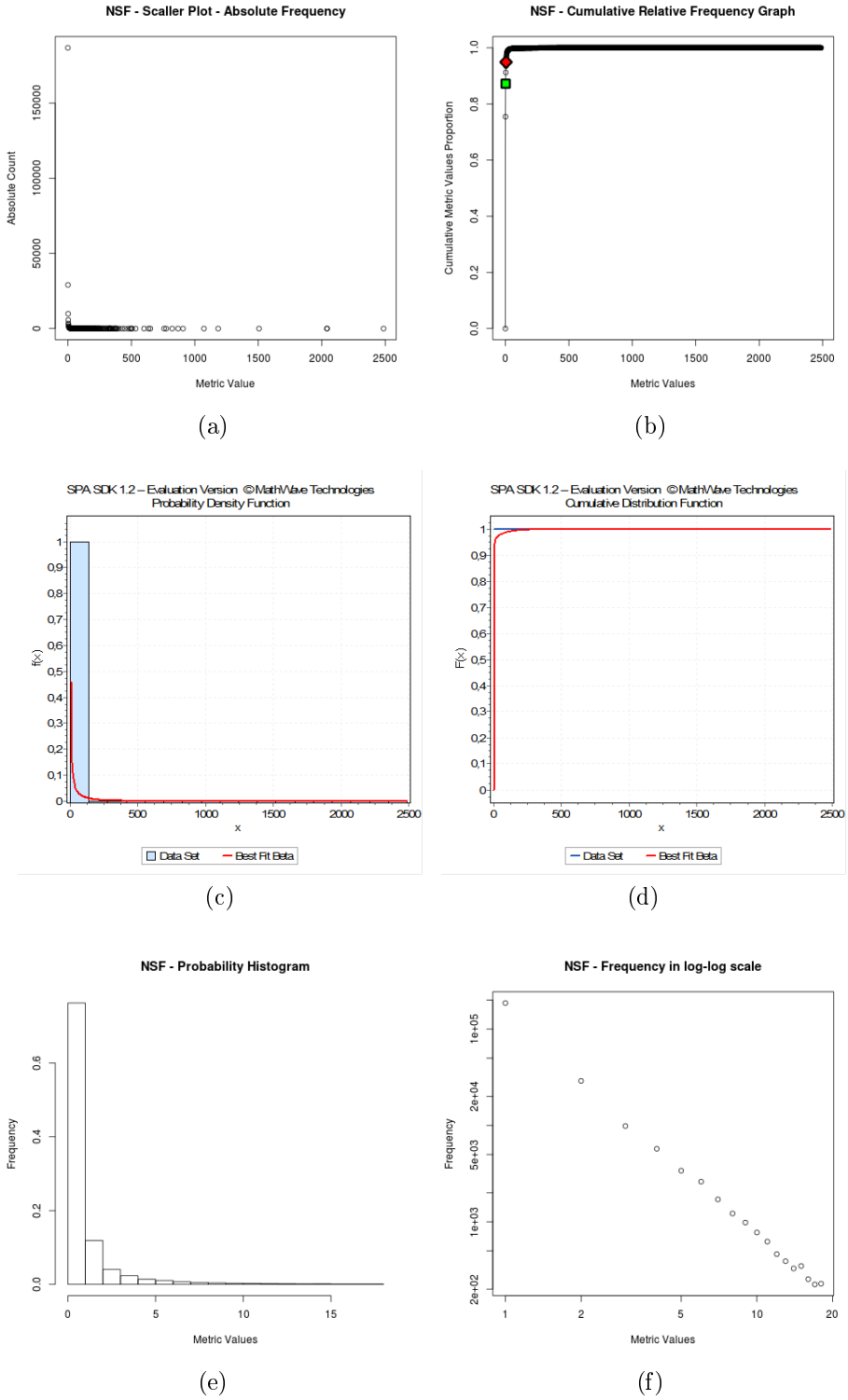


Figura A.15: Número de Atributos Estáticos: (a) Gráfico de Dispersão (b) Gráfico de Frequência Relativa Acumulada (c) Função de Densidade de Probabilidade ajustada à Distribuição Beta (d) Gráfico de Frequência Relativa Acumulada ajustado à Distribuição Beta (e) Histograma de Probabilidade (f) Histograma em escala *log-log*.

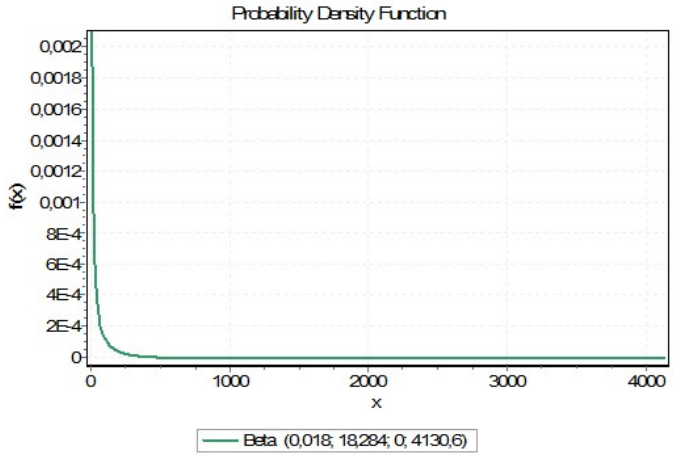


Figura A.16: NSF - Função de Densidade de Probabilidade com os parâmetros retornados pela ferramenta *EasyFit*

A.1.10 Número de Parâmetros (PAR)

O Gráfico de Dispersão dos valores coletados para a métrica Número de Parâmetros (PAR), ilustrado na Figura A.19a, sugere uma distribuição de cauda-pesada. A Figura A.19b ressalta ainda mais essa característica, pois é possível observar que a aproximação dos 100% de frequência acumulada ao longo do eixo x (valores das métricas) ocorre de uma forma drasticamente rápida, ou seja, há uma aproximação quase que instantânea dos 100% dos valores. Isso significa que há um grande número de métodos com poucos parâmetros e um pequeno número de métodos com muitos parâmetros.

Como mostrado nas Figuras A.19c e A.19d, o conjunto de valores dessa métrica possui melhor ajuste à distribuição *Gumbel Max*, com parâmetros $\sigma = 0,973$ e $\mu = 0,399$, conforme retornado pela ferramenta *EasyFit*. Essa distribuição apresenta uma curva assimétrica à direita. A Figura A.20 mostra a Função Densidade de Probabilidade com os parâmetros retornados, onde é possível visualizar a característica de assimetria à direita da distribuição com esses parâmetros.

A Figura A.19e mostra o histograma de probabilidade dos dados. Percebe-se que a segunda barra, que representa os valores de PAR igual a 1, constitui mais de 60% dos dados, enquanto a terceira barra, que representa os valores igual a 2, constitui menos de 10% dos dados. Novamente, fica ressaltada a característica da distribuição ser de cauda-pesada. A Figura A.19f mostra esses dados em escala logarítmica. Percebe-se uma reta inclinda à esquerda, característica de uma lei de potência.

Foram identificados os valores que representam o 80° e 98° percentil dos dados, que correspondem aos valores 2 e 4. Isso significa que 80% dos métodos possuem $PAR \leq 2$ e 98% dos métodos possuem $PAR \leq 4$. Os valores 80% e 98% foram identifi-

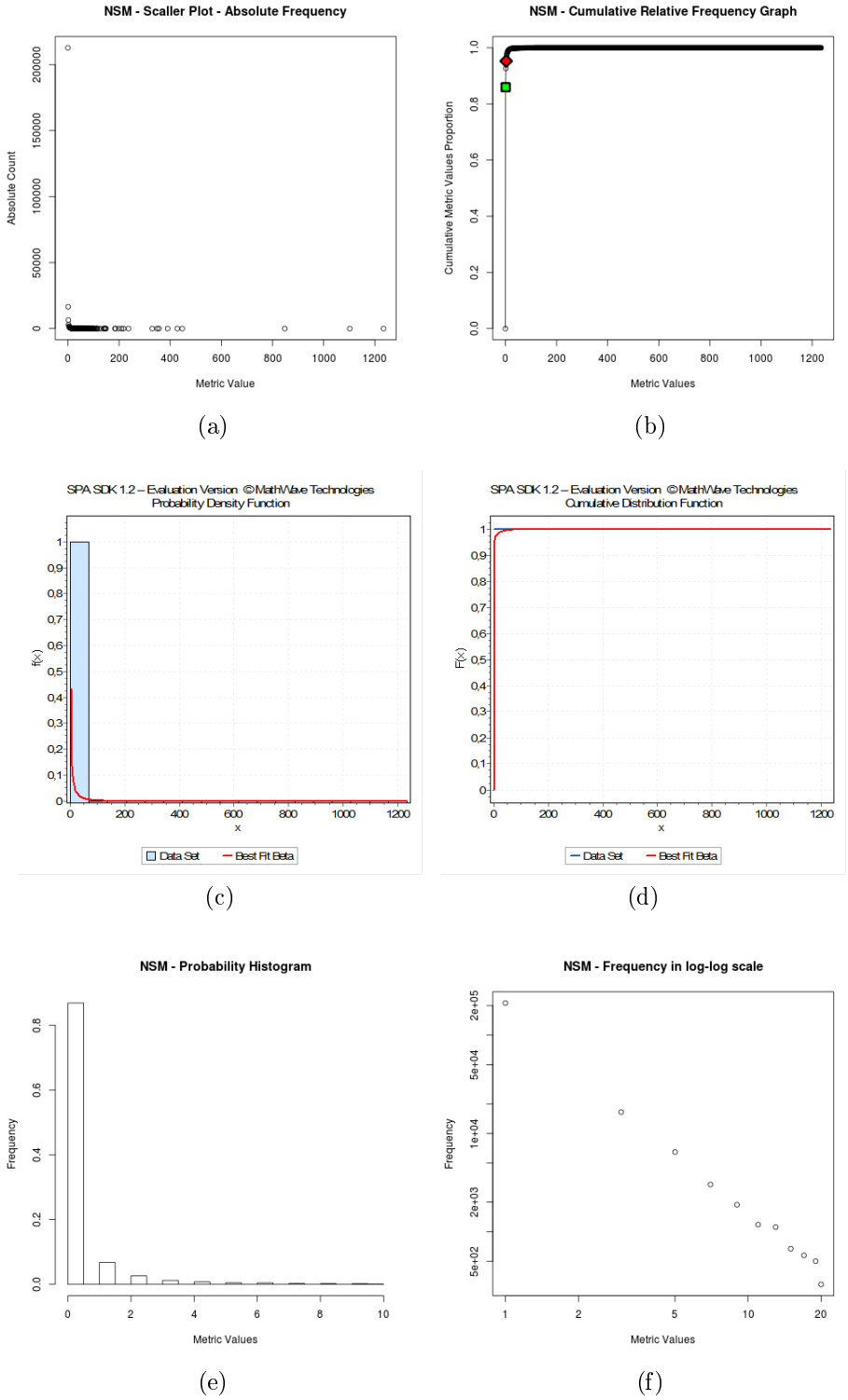


Figura A.17: Número de Métodos Estáticos: (a) Gráfico de Dispersão (b) Gráfico de Frequência Relativa Acumulada (c) Função de Densidade de Probabilidade ajustada à Distribuição Beta (d) Gráfico de Frequência Relativa Acumulada ajustado à Distribuição Beta (e) Histograma de Probabilidade (f) Histograma em escala *log-log*.

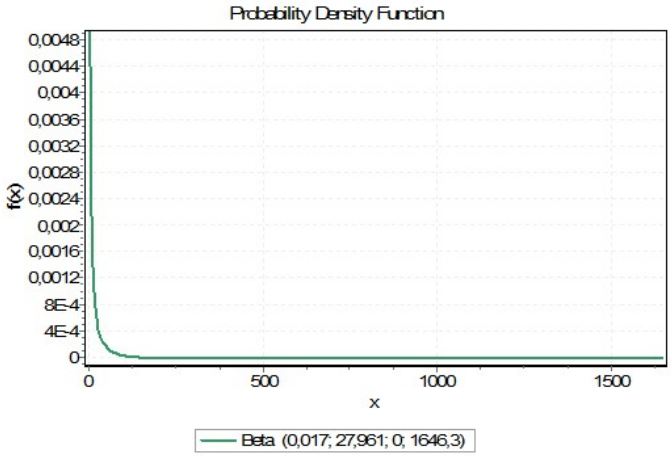


Figura A.18: NSM - Função de Densidade de Probabilidade com os parâmetros retornados pela ferramenta *EasyFit*

cados por meio de uma análise visual no Gráfico de Frequência Relativa Acumulada da Figura A.19b. Os pontos marcados com a cor verde / quadrado e vermelha / losango representam as regiões identificadas nessa análise. A Tabela A.10 sumariza os valores referência derivados para a métrica Número de Parâmetros (PAR).

Tabela A.10: Valores referência para PAR

Bom/Frequente	**Regular/Ocasional**	**Ruim/Raro**
$PAR \leq 2$	$2 < PAR \leq 4$	$PAR > 4$

A.1.11 Índice de Especialização (SIX)

O Gráfico de Dispersão dos valores coletados para a métrica Índice de Especialização (SIX), na Figura A.21a, sugere uma distribuição de cauda-pesada. A Figura A.21b ressalta ainda mais essa característica, pois nos permite observar que a aproximação dos 100% de frequência acumulada ao longo do eixo x (valores das métricas) ocorre de uma forma drasticamente rápida, ou seja, há uma aproximação quase que instantânea dos 100% dos valores. Isso significa que há um grande número de classes que pouco ou nada sobrescrevem o comportamento de suas superclasses e um número pequeno de classes que muito sobrescrevem o comportamento de suas superclasses.

Como mostrado nas Figuras A.21c e A.21d, o conjunto de valores dessa métrica possuem melhor ajuste à distribuição *Power Function*, com parâmetros $\alpha = 0,031$, $a = 0,000$ e $b = 24,578$, conforme retornado pela ferramenta *EasyFit*. Essa distribuição tem característica cauda-pesada. A Figura A.22 mostra a Função Densidade de

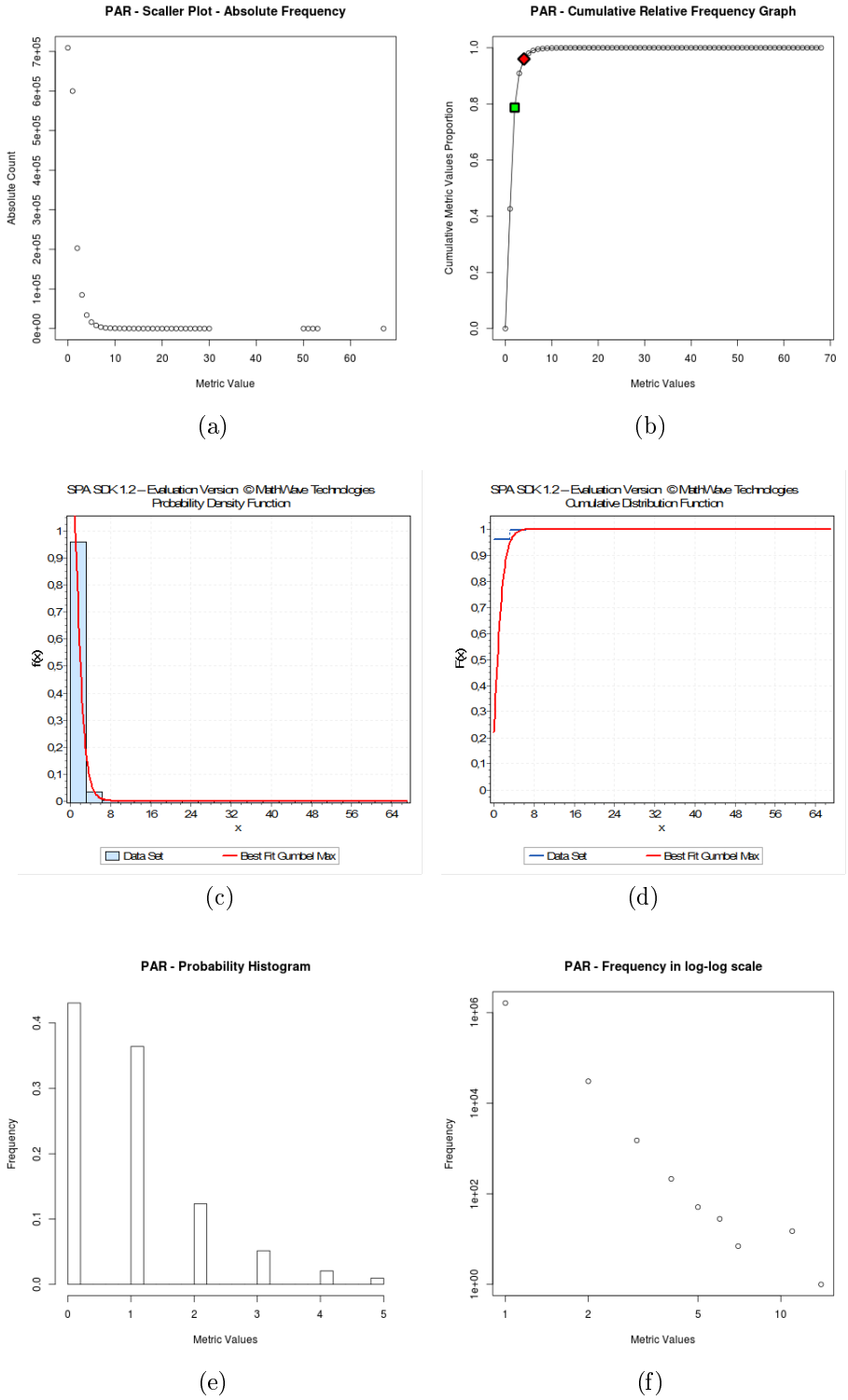


Figura A.19: Número de Parâmetros: (a) Gráfico de Dispersão (b) Gráfico de Frequência Relativa Acumulada (c) Função de Densidade de Probabilidade ajustada à Distribuição Gumbel Max (d) Gráfico de Frequência Relativa Acumulada ajustado à Distribuição Gumbel Max (e) Histograma de Probabilidade (f) Histograma em escala *log-log*.

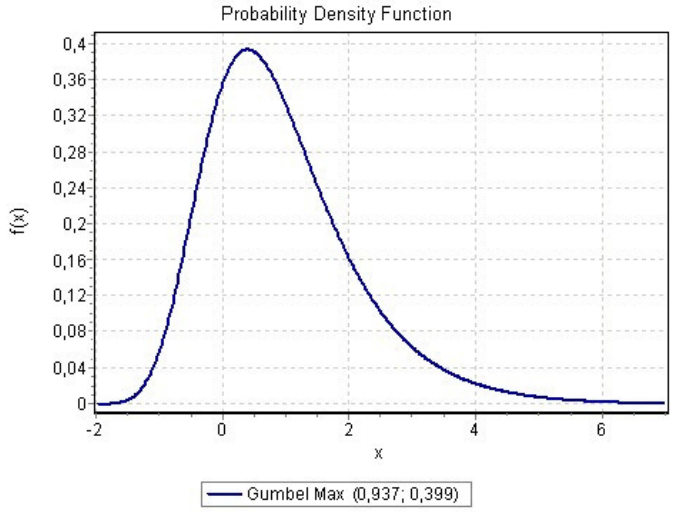


Figura A.20: PAR - Função de Densidade de Probabilidade com os parâmetros retornados pela ferramenta *EasyFit*

Probabilidade com os parâmetros retornados, onde é possível visualizar a característica de cauda-pesada da distribuição com esses parâmetros.

A Figura A.21e mostra o histograma de probabilidade dos dados. Percebe-se que a primeira barra, que representa o valor 0, constitui mais de 80% dos dados, enquanto a segunda barra, que representa o valor 1, constitui menos de 10% dos dados. Novamente, fica ressaltada a característica da distribuição ser de cauda-pesada. A Figura A.21f mostra esses dados em escala logarítmica. Percebe-se uma reta inclinada à esquerda, característica de uma lei de potência.

Foram identificados os valores que representam o 70° e 90° percentil dos dados, que correspondem aos valores 0,019 e 1,333. Isso significa que 70% das classes possuem $SIX \leq 0,019$ e 90% das classes possuem $SIX \leq 1,333$. Os valores 70% e 90% foram identificados por meio de uma análise visual no Gráfico de Frequência Relativa Acumulada da Figura A.21b. Os pontos marcados com a cor verde / quadrado e vermelha / losango representam as regiões identificadas nessa análise. A Tabela A.11 sumariza os valores referência derivados para a métrica Índice de Especialização (SIX).

Tabela A.11: Valores referência para SIX

Bom/Frequente	**Regular/Ocasional**	**Ruim/Raro**
$SIX \leq 0,019$	$0,019 < SIX \leq 1,333$	$SIX > 1,333$

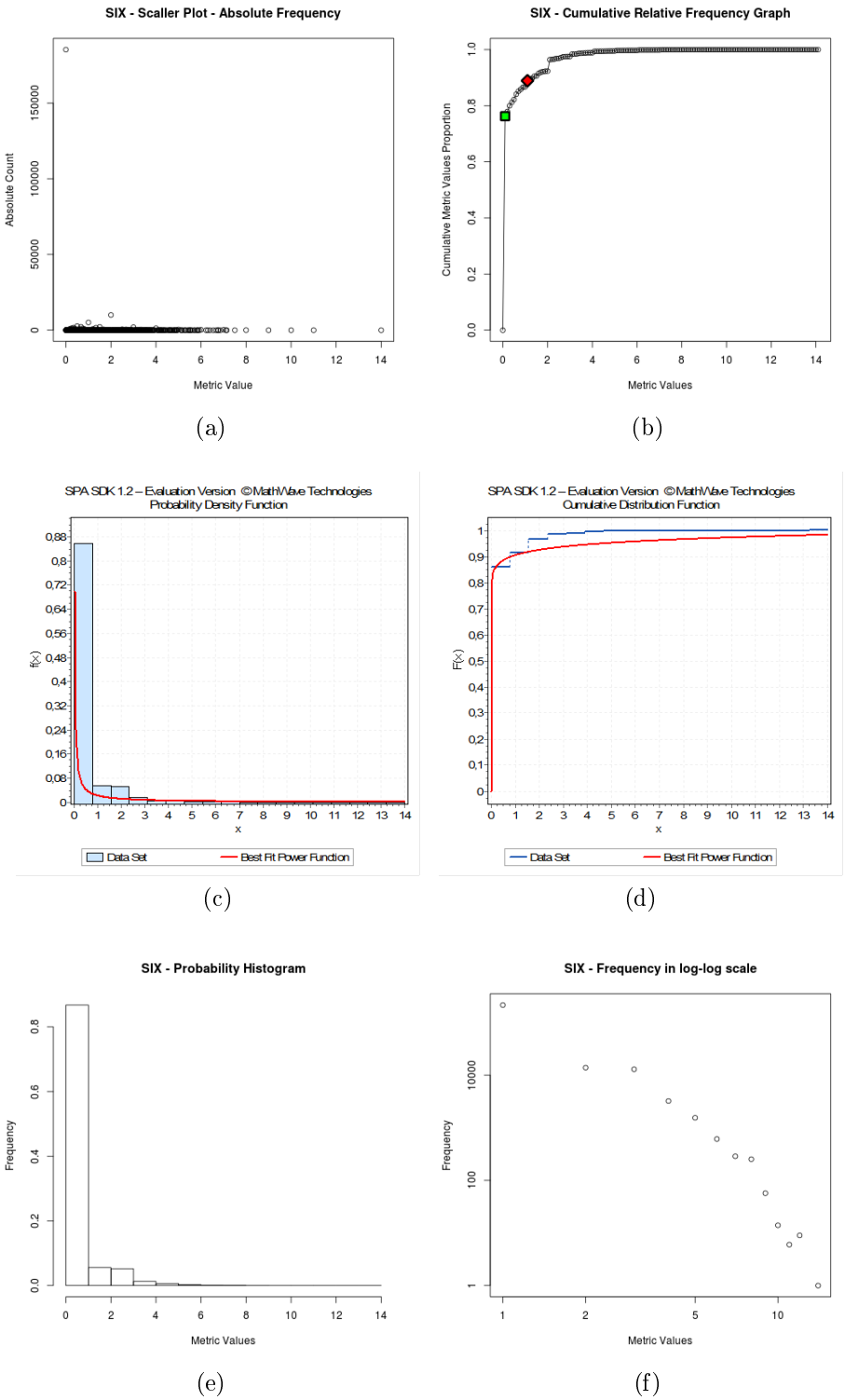


Figura A.21: Índice de Especialização: (a) Gráfico de Dispersão (b) Gráfico de Frequência Relativa Acumulada (c) Função de Densidade de Probabilidade ajustada à Distribuição Power Function (d) Gráfico de Frequência Relativa Acumulada ajustado à Distribuição Power Function (e) Histograma de Probabilidade (f) Histograma em escala *log-log*.

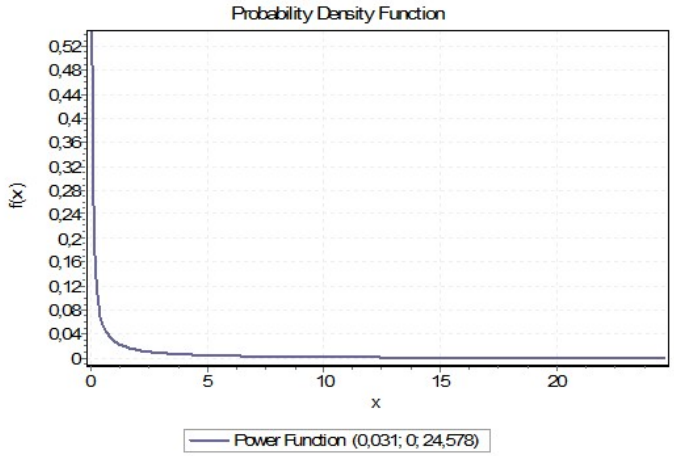


Figura A.22: SIX - Função de Densidade de Probabilidade com os parâmetros retornados pela ferramenta *EasyFit*

A.1.12 Complexidade de *McCabe* (VG)

O Gráfico de Dispersão dos valores coletados para a métrica Complexidade de *McCabe* (VG), na Figura A.23a, sugere uma distribuição de cauda-pesada. A Figura A.23b ressalta ainda mais essa característica, uma vez que é possível observar que a aproximação dos 100% de frequência acumulada ao longo do eixo x (valores das métricas) ocorre de uma forma drasticamente rápida, ou seja, há uma aproximação quase que instantânea dos 100% dos valores. Isso significa que há um grande número de métodos com poucos caminhos independetes em seu grafo de execução e poucos métodos com um grande número de caminhos independentes em seu grafo de execução.

Como mostrado nas Figuras A.23c e A.23d, o conjunto de valores dessa métrica possui melhor ajuste à distribuição *Chi-Squared*, com parâmetros $\nu = 1,000$ e $\gamma = 1,000$, conforme retornado pela ferramenta *EasyFit*. Essa distribuição tem característica cauda-pesada. A Figura A.24 mostra a Função Densidade de Probabilidade com os parâmetros retornados, onde é possível visualizar a característica de cauda-pesada da distribuição com esses parâmetros.

A Figura A.23e mostra o histograma de probabilidade dos dados. Percebe-se que a primeira barra, que representa o valor 1, constitui pouco menos de 70% dos dados, enquanto a segunda barra, que representa o valor 2, constitui pouco mais de 10% dos dados. Novamente, fica ressaltada a característica da distribuição ser de cauda-pesada. A Figura A.23f mostra esses dados em escala logarítmica. Percebe-se uma reta inclinada à esquerda, característica de uma lei de potência.

Foram identificados os valores que representam o 70° e 90° percentil dos dados, que correspondem aos valores 2 e 4. Isso significa que 70% dos métodos possuem

$VG \leq 2$ e 90% dos métodos possuem $VG \leq 4$. Os valores 70% e 90% foram identificados por meio de uma análise visual no Gráfico de Frequência Relativa Acumulada da Figura A.23b. Os pontos marcados com a cor verde / quadrado e vermelha / losango representam as regiões identificadas nessa análise. A Tabela A.12 sumariza os valores referência derivados para a métrica Complexidade de *McCabe* (VG).

Tabela A.12: Valores referência para VG

Bom/Frequente	**Regular/Ocasional**	**Ruim/Raro**
$VG \leq 2$	$2 < VG \leq 4$	$VG > 4$

.

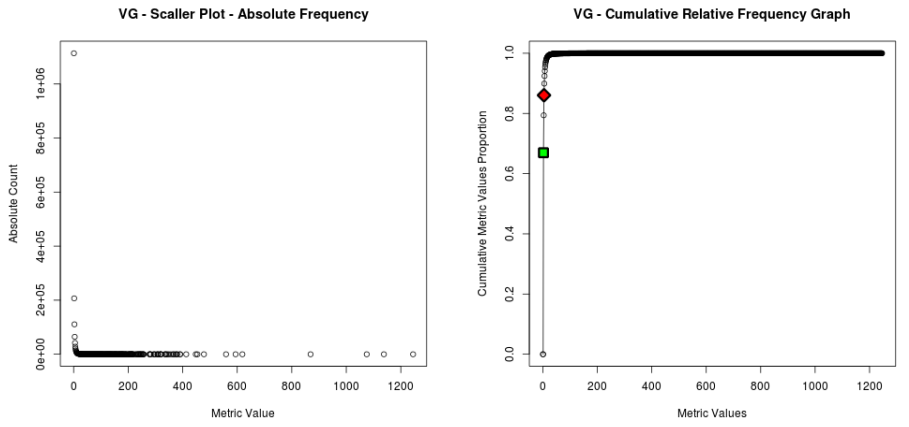
A.1.13 Métodos Ponderados por Classe (WMC)

O Gráfico de Dispersão dos valores coletados para a métrica Métodos Ponderados por Classe (WMC), na Figura A.25a, sugere uma distribuição de cauda-pesada. A Figura A.25b ressalta ainda mais essa característica, pois é possível observar que a aproximação dos 100% de frequência acumulada ao longo do eixo x (valores das métricas) ocorre de uma forma drasticamente rápida, ou seja, há uma aproximação quase que instantânea dos 100% dos valores. Isso significa que há um grande número de classes com baixo somatório das complexidades dos métodos que a constituIm e um baixo número de classes com altos valores desse somatório.

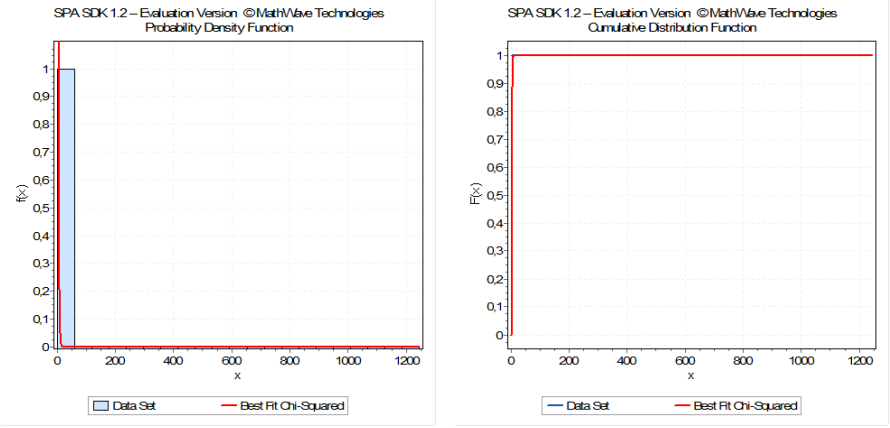
Como mostrado nas Figuras A.25c e A.25d, o conjunto de valores dessa métrica possui melhor ajuste à distribuição *Log-Logistic*, com parâmetros $\alpha = 1,142$, $\beta = 4,687$ $\gamma = 0,000$, conforme retornado pela ferramenta *EasyFit*. Essa distribuição tem característica de cauda-pesada. A Figura A.26 mostra a Função Densidade de Probabilidade com os parâmetros retornados, sendo possível visualizar a característica de cauda-pesada da distribuição com esses parâmetros.

A Figura A.25e mostra o histograma de probabilidade dos dados. Percebe-se que a primeira barra, que representa os valores entre 0 e 10, constituI pouco menos de 70% dos dados, enquanto a segunda barra, que representa os valores entre 10 e 20, constituI pouco mais de 15% dos dados. Novamente, fica ressaltada a característica da distribuição ser cauda-pesada. A Figura A.25f mostra esses dados em escala logarítmica. Percebe-se uma reta inclinada à esquerda, característica de uma lei de potência.

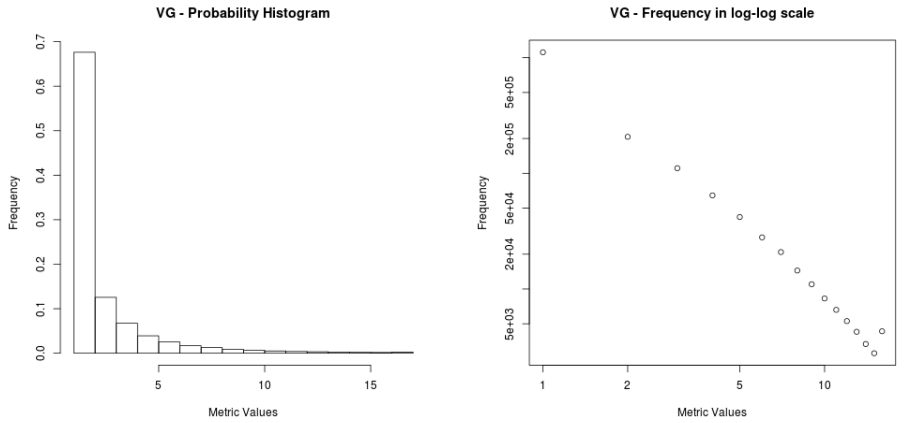
Foram identificados os valores que representam o 70° e 90° percentil dos dados, que correspondem aos valores 11 e 34. Isso significa que 70% das classes possuem $WMC \leq 11$ e 90% das classes possuem $WMC \leq 34$. Os valores 70% e 90% foram identificados por meio de uma análise visual no Gráfico de Frequência Relativa Acumulada da Figura A.25b. Os pontos marcados com a cor verde / quadrado e vermelha /



(a) (b)



(c) (d)



(e) (f)

Figura A.23: Complexidade de *McCabe*: (a) Gráfico de Dispersão (b) Gráfico de Frequência Relativa Acumulada (c) Função de Densidade de Probabilidade ajustada à Distribuição Power Function (d) Gráfico de Frequência Relativa Acumulada ajustado à Distribuição Power Function (e) Histograma de Probabilidade (f) Histograma em escala *log-log*.

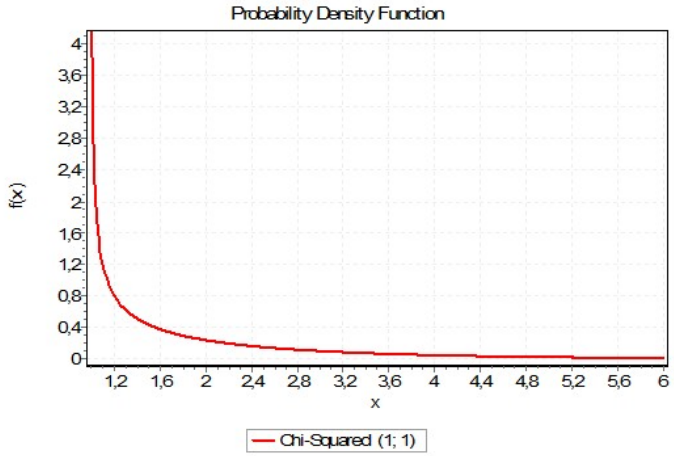


Figura A.24: VG - Função de Densidade de Probabilidade com os parâmetros retornados pela ferramenta *EasyFit*

losango representam as regiões identificadas nessa análise. A Tabela A.13 sumariza os valores referência derivados para a métrica Métodos Ponderados por Classe (WMC).

Tabela A.13: Valores referência para WMC

Bom/Frequente	**Regular/Ocasional**	**Ruim/Raro**
$WMC \leq 11$	$11 < WMC \leq 34$	$WMC > 34$

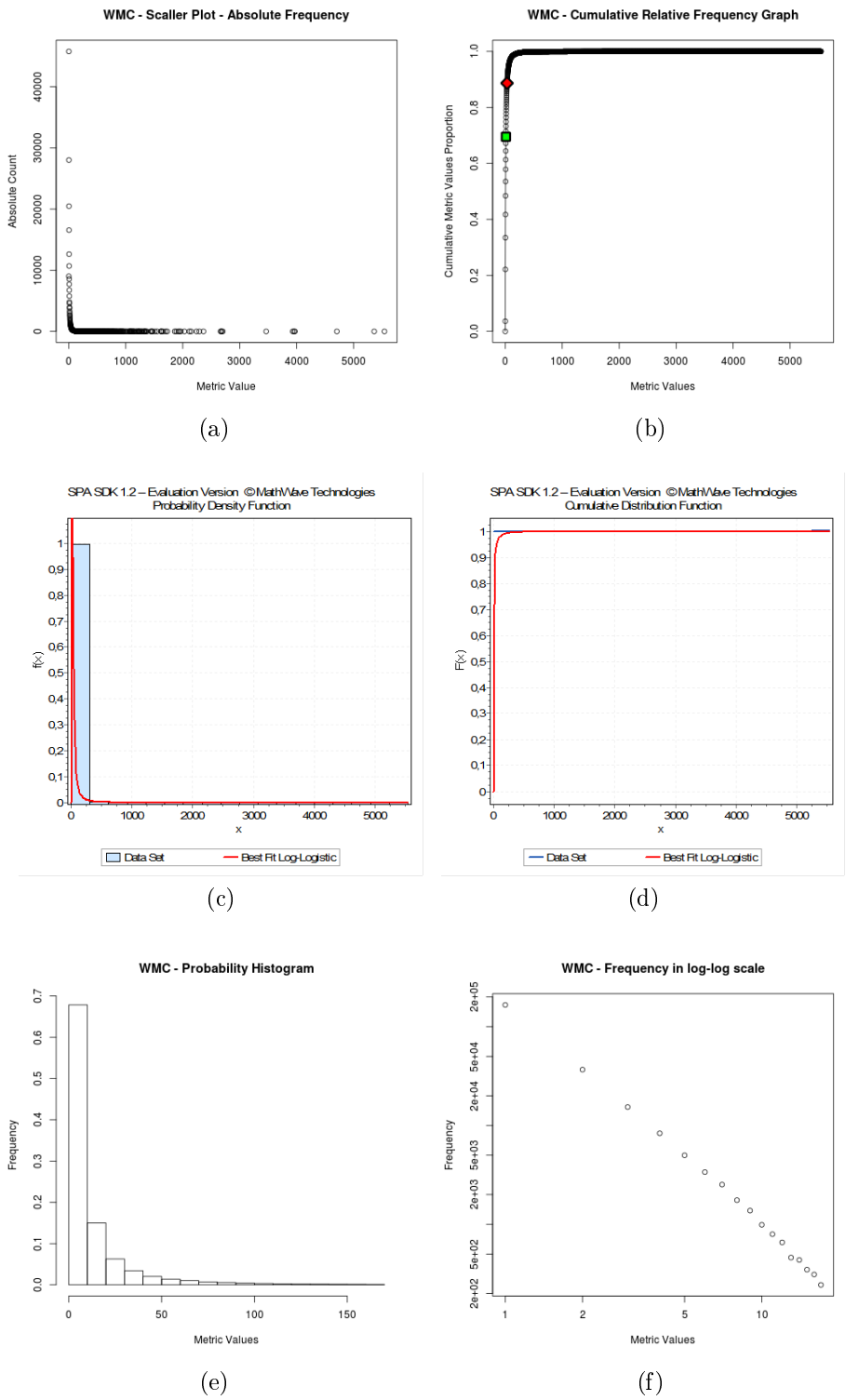


Figura A.25: Métodos Ponderados por Classe: (a) Gráfico de Dispersão (b) Gráfico de Frequência Relativa Acumulada (c) Função de Densidade de Probabilidade ajustada à Distribuição Log-Logistic (d) Gráfico de Frequência Relativa Acumulada ajustado à Distribuição Log-Logistic (e) Histograma de Probabilidade (f) Histograma em escala *log-log*.

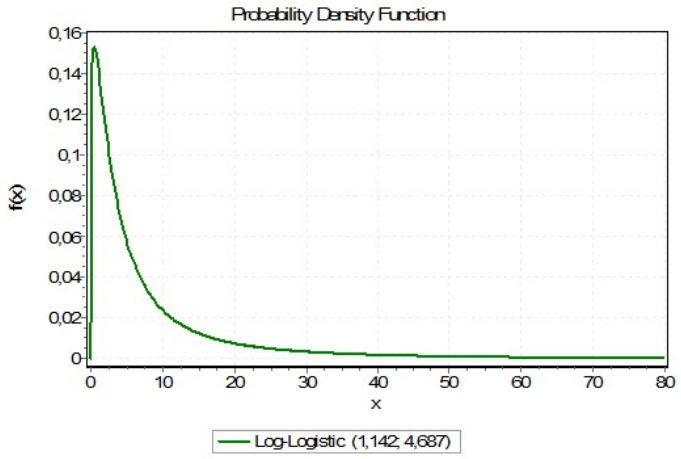


Figura A.26: WMC - Função de Densidade de Probabilidade com os parâmetros retornados pela ferramenta *EasyFit*