MARIANA DE AZEVEDO SANTOS

UM ESTUDO QUANTITATIVO PARA CARACTERIZAÇÃO DA QUALIDADE INTERNA DE SISTEMAS DE SOFTWARE ORIENTADOS A OBJETOS *OPEN-SOURCE*

LAVRAS-MG

2015

MARIANA DE AZEVEDO SANTOS

UM ESTUDO QUANTITATIVO PARA CARACTERIZAÇÃO DA QUALIDADE INTERNA DE SISTEMAS DE SOFTWARE ORIENTADOS A OBJETOS *OPEN-SOURCE*

Dissertação apresentada à Universidade Federal de Lavras, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, área de concentração em Engenharia de Software e Banco de dados, para a obtenção do título de Mestre.

Orientador

Dr. Heitor Augustus Xavier Costa

Coorientador

Dr. Paulo Henrique de Souza Bermejo

LAVRAS-MG

2015

Ficha catalográfica elaborada pelo Sistema de Geração de Ficha Catalográfica da Biblioteca Universitária da UFLA, com dados informados pelo(a) próprio(a) autor(a).

Santos, Mariana de Azevedo.
Um estudo quantitativo para caracterização da qualidade interna de sistemas de software orientados a objetos *Open-Source* / Mariana de Azevedo Santos. – Lavras : UFLA, 2015.
214 p. : il.

Dissertação (mestrado acadêmico)–Universidade Federal de Lavras, 2015.
Orientador: Heitor Augustus Xavier Costa.
Bibliografia.

1. Qualidade de Software. 2. Modelo de Equações Estruturais. 3. Qualidade Interna. I. Universidade Federal de Lavras. II. Título.

MARIANA DE AZEVEDO SANTOS

UM ESTUDO QUANTITATIVO PARA CARACTERIZAÇÃO DA QUALIDADE INTERNA DE SISTEMAS DE SOFTWARE ORIENTADOS A OBJETOS *OPEN-SOURCE*

Dissertação apresentada à Universidade Federal de Lavras, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, área de concentração em Engenharia de Software e Banco de dados, para a obtenção do título de Mestre.

APROVADA em 27 de abril de 2015.

Dr. Paulo Henrique de Souza Bermejo	UFLA/DCC
Dr. Marcelo Silva de Oliveira	UFLA/DEX
Dr. Marco Aurélio Gerosa	USP/IME

Dr. Heitor Augustus Xavier Costa
Orientador

LAVRAS-MG

2015

AGRADECIMENTOS

Aos meus pais, Paulo e Leila, pelo amor, dedicação, incentivo e paciência sempre. Pelo exemplo de luta e de coragem, que me fizeram persistir no caminho escolhido, mesmo diante das dificuldades encontradas.

Ao meu irmão Lucas, pelo companheirismo, amizade, carinho e atenção. À minha família, vovó Anália, tias e tios queridos, primos, primas e cunhada, pelo apoio e carinho.

À Universidade Federal de Lavras (UFLA) e aos funcionários e professores do Departamento de Ciência da Computação (DCC), pela oportunidade concedida para a realização do mestrado, pelos ensinamentos transmitidos e pelo apoio nesta conquista. Em especial, ao professor Dr. André Vital Saúde, pelos conselhos e pela amizade.

Aos professores Dr. Heitor Augustus Xavier Costa e Dr. Paulo Henrique de Souza Bermejo, pela orientação e dedicação neste período de trabalho. Sou grata por todo ensinamento que me foi passado, e tenham certeza de que a realização deste trabalho com vocês me proporcionou um grande crescimento pessoal e profissional.

Aos professores Dr. Marcelo Silva de Oliveira e Dr. André Pimenta Freire, que acompanham o trabalho desde o período da qualificação do projeto, pela disposição em participar como membros da banca examinadora, fornecendo preciosas dicas para que o trabalho fosse construído até aqui. Ao professor Dr. Marco Aurélio Gerosa, pela atenção ao trabalho realizado e pela disposição em participar da banca examinadora.

Aos amigos do PPGCC/UFLA, por esses dois anos de alegrias, angústias, expectativas, muito estudo, troca de conhecimentos e experiências.

Aos amigos da *Mitah Technologies*, meus colegas de profissão, por esses dois anos de paciência diante das ausências temporárias na empresa e pelo companheirismo sempre.

Aos amigos de tantos anos, por sempre me apoiarem incondicionalmente em todos os momentos.

Muito obrigada!

RESUMO

Organizações desenvolvedoras de software estão cada vez mais preocupadas com a garantia da qualidade de sistemas de software, uma vez que esses sistemas precisam ser fáceis de evoluir e de manter. Entretanto, apesar de serem necessárias atividades relacionadas à garantia da qualidade e manutenção em sistemas, essas atividades são consideradas as mais longas e complexas do ciclo de vida do software. Aproveitando a tendência crescente e os benefícios advindos da iniciativa *open-source* (OS), pesquisas sobre a qualidade e a manutenibilidade de sistemas de software OS têm ganhado renovado interesse. Nesse contexto, o uso de técnicas estatísticas robustas, como PLS-SEM, para investigar e validar empiricamente modelos de qualidade de software, tem se mostrado uma alternativa eficiente para obter informações sobre a qualidade em software OS. Neste trabalho, o objetivo foi avaliar a qualidade interna de sistemas de software a partir da caracterização entre domínios nesses sistemas Java *open-source*. Os resultados do estudo indicam que: i) há domínios que possuem características similares entre si; e ii) quatro fatores podem influenciar a qualidade interna de software orientados a objetos (OO) OS quanto à manutenibilidade. Esses fatores são a Redução da Complexidade, Coesão Normalizada, Coesão não Normalizada e Aumento do Grau de Modularidade. Um modelo conceitual foi proposto para explicar a relação entre os atributos que compõem esses fatores e; iii) medidas que podem ser mais relevantes para caracterizar a manutenibilidade de sistemas de software OO OS, como por exemplo, *Fan-out* (FOUT), *Lack of Cohesion of Methods* 2 (LCOM2), *Response for Class* (RFC), *Tight Class Cohesion* (TCC), *Loose Class Cohesion* (LCC). Os resultados deste estudo contribuem para auxiliar profissionais da área Engenharia de Software, na construção de sistemas de alta qualidade, com baixo custo em manutenção, que atendam os padrões estabelecidos e as necessidades dos usuários.

Palavras-chave: Qualidade de software. Software *open-source*. Orientação a objetos. Medidas de software. PLS-SEM.

ABSTRACT

Software development organizations are increasingly concerned with software quality assurance, given that these systems must be easily developed and maintained. However, despite necessity, activities regarding quality assurance and maintenance of software systems are considered the longest and most complex in software development lifecycle. In addition, if productivity in the development phase is low, the individuals involved in the software development process may have to invest a lot of time in post-development maintenance activities. Taking advantage of this growing trend and of the benefits obtained from open-source initiative, such as extensive knowledge exchange, automated large-scale analysis and ability to quickly drive innovations, researches on OS software quality and maintainability have gained renewed interest. The use of robust statistical techniques, such as PLS-SEM, to investigate and empirically validate software quality models has also been an efficient alternative to obtain information on OS software quality. The aim of this study was to evaluate the internal software quality by means of characterizing software domain in Java open-source systems. The study results indicate that there are: i) domains with similar traits to each other; and ii) four factors can influence the internal quality of OO software to present better maintainability, being them Reduced Complexity, Normalized Cohesion, Non-normalized Cohesion and Increase of Modularity Degree. We propose a conceptual model to explain the relationship between the internal software quality attributes (software measures) in these factors. The results also indicate the existence of measures that can be most relevant in characterizing maintainability in OO OS software systems, such as *Fan-out* (FOUT), *Lack of Cohesion of Methods* 2 (LCOM2), *Response for Class* (RFC), *Tight Class Cohesion* (TCC), *Loose Class Cohesion* (LCC). Thus, this study aids software engineering professionals, allowing the development of high quality and low maintenance cost software that meet the established standards and the user needs.

Keywords: Software quality. Open-source software. Object-oriented. Software measures. PLS-SEM.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

A demanda por qualidade é intensificada pela dependência cada vez maior da sociedade por sistemas de software (KAN, 2002; PRESSMAN, 2014). Quando no processo de desenvolvimento de sistemas de software são introduzidos erros na funcionalidade desses sistemas (por exemplo, um sistema financeiro com problemas na função de faturamento), tais erros podem ser rastreados como um problema de qualidade de software. Nesse sentido, a qualidade tem sido trazida para o centro do processo de desenvolvimento de software, pois não é mais vista como um fator de vantagem competitiva; tornou-se uma condição necessária, se uma empresa deseja desenvolver software de alta qualidade, a baixos custos e dentro dos prazos estipulados (KAN, 2002).

Embora a qualidade do software seja um fator crítico para um software no mercado ter bom faturamento e grande adesão de usuários, o seu conceito é complexo de definir, de descrever, de compreender e de mensurar (KITCHENHAM; WALKER, 1989). Qualidade de software é um conceito multidimensional e, como em qualquer conceito, há níveis de abstração, podendo ser conceituada de forma ampla ou específica. Além disso, o conceito de qualidade é popular e da linguagem cotidiana, que pode ter significados diferentes, quando referida em termos pessoais e profissionais (KAN, 2002).

No contexto de software, a qualidade pode ser definida como o grau em que as características de um produto/serviço satisfazem as necessidades **explícitas**[1] e **implícitas**[2] dos *stakeholders* (partes/pessoas interessadas por alguma razão no desenvolvimento e no uso do software) agregando valor a esse

---

[1] Necessidades explícitas são as condições e os objetivos propostos por aqueles que produzem o software.

[2] Necessidades implícitas são necessidades subjetivas dos usuários que se tornam evidentes quando o sistema de software é usado em condições especiais

produto/serviço (INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 2011). Por exemplo, a qualidade de projeto é alcançada se o projeto completo está conforme os requisitos estabelecidos pelo proprietário, pelo projetista e pelo cliente. Do ponto de vista do proprietário, a qualidade é caracterizada pelo atendimento de requisitos (BUBSHAIT, 1994), por exemplo: i) funcionalidade e estética; ii) realizado no prazo e no custo estimados; iii) custo do ciclo de vida; iv) operabilidade e manutenibilidade; e v) requisitos do ambiente e de segurança. O gerenciamento da qualidade do projeto está entre as melhores práticas para a gestão de projetos (PMBOK GUIDE, 2013).

Especificamente, a norma ISO/IEC 9126-1 detalha o conceito de qualidade de software em outros três subconceitos (INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 2001): i) **qualidade externa**, que garante a conformidade do produto com as necessidades do usuário; ii) **qualidade em uso**, que representa o quão os usuários podem atingir seus objetivos em determinado ambiente, mas não as propriedades do software em si e; iii) **qualidade interna**, que corresponde à qualidade da organização do código ou de sua complexidade. Na norma ISO/IEC 25010, os modelos de qualidade externa e interna foram reunidos em um único modelo chamado "modelo de qualidade de produto" (*product quality model*) e o modelo de qualidade em uso sofreu alterações, tais como: i) alteração do nome de algumas características, por exemplo, **Funcionalidade**, presente na norma ISO/IEC 9126-1, para **Adequação Funcional** (INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 2011); ii) a subcaracterística **Completude Funcional** está presente na ISO/IEC 25010, mas não existe na norma ISO/IEC 9126-1 e; iii) a subcaracterística **Segurança de Acesso** na norma ISO/IEC 9126-1 foi transformada em característica na norma ISO/IEC 25010. O modelo de qualidade de produto é constituído por oito características (INTERNATIONAL

ORGANIZATION FOR STANDARDIZATION, 2011): i) **Adequação Funcional**; ii) **Compatibilidade**; iii) **Usabilidade**; iv) **Confiabilidade**; v) **Segurança**; vi) **Manutenibilidade**; vii) **Portabilidade** e; viii) **Eficiência do Desempenho**. Neste trabalho foi utilizado o modelo de qualidade do produto para a avaliação da qualidade interna de sistemas de software. Mais precisamente, essa avaliação foi executada sob a característica de qualidade Manutenibilidade.

A qualidade interna do código (o quão bem escrito ele é) influencia a qualidade de sistemas de software (PLOSCH et al., 2007). Todavia, apesar de nem sempre ser importante para o usuário final (não são percebidas diretamente), pode afetar sua percepção de qualidade desses sistemas de forma indireta, por exemplo, no tempo de manutenção ou de evolução do software promovido pela equipe de desenvolvimento (BAGGEN et al., 2012; PLOSCH et al., 2007). A medição interna da qualidade de um sistema de software refere-se ao grau em que um conjunto de atributos estáticos satisfaz as necessidades explícitas e implícitas para ser utilizado em condições específicas (INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 2011). Por atributo define-se como propriedade inerente ou característica de uma entidade que pode ser distinguida quantitativa ou qualitativamente de forma humana ou automática (INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 2007). No contexto da qualidade software, as variáveis pelas quais se atribui um valor como resultado de uma medição desses atributos, capazes de caracterizar, de avaliar e de prever a qualidade de processos, de projetos e de sistemas de software são conhecidas como medidas de software (INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 2010; KAN, 2002; KAYARVIZHY; KANMANI, 2011).

Medida é uma escala utilizada para a categorização e a medição de dados qualitativos de um sistema de software ou de suas especificações internas

e externas (INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 2011). Medidas desses sistemas são propostas por pesquisadores, pela indústria e por grupos de trabalhos relacionados a organizações nacionais e internacionais. Por exemplo, têm-se as medidas publicadas nas normas ISO/IEC 9126 e ISO/IEC 25000. Elas podem ser coletadas ao longo do processo de desenvolvimento de software identificando aspectos de um projeto, tais como, o esforço, o custo, a complexidade e a modularidade. Quando medidas são específicas, elas representam estruturas inerentes à tecnologia de programação adotada, por exemplo, herança, na tecnologia de orientação a objetos (OO) (TIAN; CHEN; ZHANG, 2008). Além disso, medidas que podem ser aplicadas para a avaliação de características de qualidade específicas, como a manutenibilidade.

A manutenibilidade, definida como a facilidade em realizar manutenção em um sistema de software (INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 2011), é fator crítico de qualidade para esse sistema continar a ser utilizado por seus usuários (PING, 2010; SUNDAY, 1989). Com isso, um sistema de software manutenível deve obedecer a alguns conceitos de desenvolvimento da Engenharia de Software e da Arquitetura de Software abordados neste trabalho. Dessa forma, pode-se dizer que a implementação de modelos, de práticas e de diretrizes para manutenção de um sistema de software de qualidade, promove a redução de esforços na evolução desses sistemas, assegurando, por exemplo, extensibilidade e legibilidade (PING, 2010; PRESSMAN, 2014; SOMMERVILLE, 2015).

1.1 Motivação

Sistemas de software precisam evoluir continuamente durante o seu ciclo de vida (BADRI; DROUIN; TOURE, 2012). Nesse sentido, organizações

do ramo de desenvolvimento de sistemas de software estão cada vez mais preocupadas com a manutenção e a garantia da qualidade desses sistemas. Dentre essas preocupações está o desenvolvimento, de forma eficiente, de sistemas a baixos custos e de alta qualidade para atender as necessidades dos usuários e continuar a ser úteis (BADRI; DROUIN; TOURE, 2012; HAMID; HASAN, 2010; KAN, 2002; SINGH; KANNOJIA, 2013).

Entretanto, apesar de serem necessárias atividades relacionadas à garantia da qualidade e à manutenção em projetos de software, essas atividades são consideradas as mais longas e complexas do ciclo de vida do software (PRESSMAN, 2014). Por exemplo, se a produtividade no processo de desenvolvimento é baixa, as pessoas envolvidas nesse processo podem ter que investir mais tempo em manutenção pós-desenvolvimento (GRANJA-ALVAREZ; BARRANCO-GARCÍA, 1997). Apesar da complexidade envolvida nessas atividades, a garantia da qualidade desses sistemas é um componente importante no seu desenvolvimento, seja o software proprietário ou *open-source* (OS) (SELIYA; KHOSHGOFTAAR, 2007).

O modelo de software *open-source* tem despertado interesse e suscitado reflexões nos mais diversos âmbitos (governo, academia, empresas etc), no Brasil e no exterior (SOFTEX, 2005). Há relatos em pesquisas de que sistemas de software comerciais estão sendo liberados sob licença OS (SOFTEX, 2005; VON HIPPEL; VON KROGH, 2003). Com a tendência crescente e os benefícios advindos dessa iniciativa, pesquisas sobre a qualidade de sistemas de software OS têm ganhado renovado interesse (BRIAND et al., 2000; SOUZA; MAIA, 2013; ZHONG; YANG; KEUNG, 2012). Esses sistemas são disponibilizados em repositórios, tais como, Github[3] e Sourceforge[4], e possuem ampla gama de domínios de software, como negócios, desenvolvimento

[3] https://github.com/
[4] http://sourceforge.net/

(navegadores *web*, IDE's, etc.) e educação (MIDHA; BHATTACHERJEE, 2012). Dentre os benefícios citados por autores de trabalhos na área (BRIAND et al., 2000; CHELIOTIS, 2009; MIDHA; BHATTACHERJEE, 2012; SOUZA; MAIA, 2013; ZHONG; YANG; KEUNG, 2012), estão a intensa troca de conhecimentos, a análise automatizada em larga escala e a capacidade de conduzir rapidamente ideias inovativas.

Entretanto, algumas dificuldades podem ser encontradas no desenvolvimento de sistemas de software OS. Esses sistemas são frequentemente desenvolvidos com modelos de ciclo de vida de desenvolvimento e gestão de diferente dos sistemas proprietários (MIDHA; BHATTACHERJEE, 2012; ZHONG; YANG; KEUNG, 2012). Na literatura existem abordagens que tentam explicar o desenvolvimento de software OS, porém a maioria dos modelos é baseada nas experiências e nos requisitos específicos de cada projeto (SAINI; KAUR, 2014). Por exemplo, um estudo propõe que o modelo do ciclo de vida do desenvolvimento de software OS consiste em três fases (CAPILUPPI; MICHLMAYR, 2007): i) **Catedral**, com desenvolvimento fechado e focado na concepção e no desenvolvimento da ideia pelo idealizador; ii) **Transição**, com protótipo estável e pronto para ser lançado pelo desenvolvedor; e iii) **Bazar**, fase em que a comunidade interessada se envolve no processo de evolução e de manutenção do software. Outro estudo propõe um modelo que abrange as etapas de planejamento do projeto, o desenvolvimento do código até a sua instalação e configuração no usuário final (SCACCHI, 2002; SCACCHI et al., 2006). Dessa forma, ainda não há um modelo padrão aplicável de forma genérica ao desenvolvimento de software OS. Além disso, um grande desafio para esses sistemas é proporcionar formas de apoiar novos desenvolvedores e a sua relação com os desenvolvedores mais antigos (STEINMACHER et al., 2014). Há relatos de que novos desenvolvedores são geralmente deixados para aprender sobre o projeto e as

técnicas utilizadas por conta própria (SCACCHI, 2002; STEINMACHER et al., 2014). Dessa forma, a qualidade do código produzido deve ser estudada e avaliada para garantir que esses sistemas sejam manuteníveis e reutilizáveis.

Nesse contexto, modelos de qualidade de software criados, utilizando técnicas estatísticas, podem ser alternativas eficientes para obter informações sobre a qualidade de sistemas de software OS (SELIYA; KHOSHGOFTAAR, 2007). Os modelos de qualidade de software são definidos como um conjunto de características que provê um *framework* com requisitos de qualidade e meios de avaliar o software (INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 2011). Esses modelos podem ser utilizados para identificar sistemas de software propensos a falhas e defeitos ou baixa manutenibilidade, modularidade e reusabilidade.

Na literatura, há trabalhos que investigam como medidas de software podem (i) prever a propensão à falhas em classes (BRIAND et al., 2000) e outras características, tais como, modularidade (KAYARVIZHY; KANMANI, 2011) e manutenibilidade (AL DALLAL, 2013; ANDA, 2007); (ii) na criação de modelos de qualidade para avaliação da aquisição de sistemas de software OS (SOTO; CIOLKOWSKI, 2009), para definir/aplicar tecnologias, processos e políticas de uso em sistemas de software OS (QUALIPSO, 2014) e (iii) para descrever características de modularidade considerando vários domínios (SOUZA; MAIA, 2013). Entretanto, esses estudos têm como limitações, a quantidade reduzida de sistemas (AL DALLAL, 2013; ANDA, 2007; BRIAND et al., 2000; JUNG; KIM; CHUNG, 2004; KAYARVIZHY; KANMANI, 2011), não há consistência entre os diferentes tipos de software e os aspectos da OO. Além disso, os pesquisadores executam análises estatísticas pouco robustas (de caráter não confirmatório) analisando a modularidade com objetivos descritivos, como a descrição das características entre domínios e a proposta de valores de referência (SOUZA; MAIA, 2013).

Diante do exposto considera-se que a falta de um modelo conceitual para caracterizar a qualidade interna de sistemas de software OS, pode dificultar a identificação de problemas de qualidade desses sistemas. Em uma pesquisa (KAYARVIZHY; KANMANI, 2011), têm-se informações de que há necessidade do mapeamento de características de qualidade de software OO, desenvolvidos em linguagens diferentes de C++, por exemplo, Java. Além disso, há informações em estudos de Engenharia de Software, sobre a consideração do domínio em que um software está inserido, ao analisar atributos de qualidade de um sistema de software. Em alguns estudos (AL DALLAL; BRIAND, 2010; ENGLISH; BUCKLEY; CAHILL, 2010; ETZKORN et al., 2004; FERREIRA et al., 2012; KANMANI et al., 2007; RATHORE; GUPTA, 2012), os autores afirmam que o resultado de medidas pode variar em sistemas de domínios diferentes. Em outros estudos (JUNG; KIM; CHUNG, 2004; MCMILLAN et al., 2011; SOUZA; MAIA, 2013), há a afirmação que o domínio de um software pode ser um fator importante no auxílio às partes interessadas na identificação de necessidades específicas e na previsão de problemas de manutenção. Essas informações sugerem um campo aberto para estudos, principalmente empíricos, que objetivam:

a) Avaliar a qualidade interna de sistemas de software, considerando os domínios nos quais estão inseridos;
b) Avaliar a qualidade interna de sistemas sob a característica Manutenibilidade, considerando a amplitude das medidas, suas propriedades e sua relevância;
c) Mapear e sintetizar essas características em modelos, para determinar se medidas são capazes de predizer e de caracterizar a qualidade de software OO, visto que os modelos propostos não são

generalizáveis e não abrangem todas as medidas e propriedades existentes;

d) Mensurar e estimar valores de referência para domínios diferentes, considerando medidas diferentes;
e) Verificar similaridades entre características de sistemas de diferentes domínios. O pressuposto subjacente é os componentes de um software de um domínio com modelos de fatores semelhantes terem características de qualidade semelhantes (SELIYA; KHOSHGOFTAAR, 2007; SOUZA; MAIA, 2013).

Criado a partir da avaliação de sistemas de software Java e validado por técnicas estatísticas confirmatórias, sobre diferentes propriedades e considerando diferentes domínios, os modelos confirmatórios permitem um estudo estatístico do ajuste entre os argumentos teóricos e os dados empíricos. Aliados a essas necessidades, sistemas de software OO geralmente contêm grande quantidade de atributos, que podem fornecer descrições sobre sua natureza e sua estrutura interna.

1.2 Objetivos

Neste trabalho, o objetivo foi avaliar a qualidade interna de software a partir da caracterização entre domínios em sistemas de software Java *open-source*. Para alcançar esse objetivo, foram considerados os seguintes objetivos específicos:

a) Identificar quais medidas de software aplicadas à tecnologia OO são mais estudadas para a caracterização da qualidade interna de sistemas de software, quanto à característica Manutenibilidade;

b) Desenvolver um *plug-in* do Eclipse IDE para coleta e medição da qualidade interna de sistemas de software;
c) Identificar fatores que caracterizam a qualidade interna de sistemas de software OO OS, quanto à sua manutenibilidade, com base nos resultados obtidos nas etapas exploratórias e na avaliação do estudo;
d) Identificar o relacionamento entre fatores em termos teóricos e verificar a correspondência estatística dos dados coletados em sistemas de software OO OS.

1.3 Estrutura do trabalho

O restante deste trabalho está organizado da seguinte forma.

A metodologia de desenvolvimento do trabalho é apresentada na Seção 2.

Conceitos sobre qualidade de software, em seus diferentes tipos (produto, processo e projeto) são abordados. Além disso, a importância da qualidade no contexto do desenvolvimento de software, as formas de avaliá-la e as medidas de software utilizadas para caracterizar sistemas de software orientados a objetos são apresentadas na Seção 3. Nesse capítulo, também são abordados conceitos de complexidade, de acoplamento e de coesão inerentes à tecnologia de orientação a objetos, por exemplo, a herança, que servem para a caracterização das propriedades das medidas utilizadas neste trabalho.

Conceitos sobre as técnicas estatísticas Análise Fatorial Exploratória (AFE) e PLS-SEM (*Partial Least Squares Structural Equation Modeling*) utilizadas são apresentadas na Seção 4.

O estado da arte do tema estudado, com a execução de uma Revisão Sistemática de Literatura sobre os trabalhos relacionados à proposta é apresentado na Seção 5.

A ferramenta computacional *O3SMeasures*, que foi desenvolvida para a coleta de dados em sitemas de software orientados a objetos *open-source* desenvolvidos em Java é apresentada na Seção 6.

A descrição e a sumarização dos dados utilizados para a obtenção do modelo multivariado, considerando os domínios dos sistemas de software selecionados são apresentadas na Seção 7.

Fatores que influenciam a qualidade de software, em relação à característica Manutenibilidade em um estudo aplicando a AFE são apresentados na Seção 8.

O modelo multivariado, cujas variáveis são os fatores encontrados na AFE, descrito e avaliado utilizando a técnica PLS-SEM é apresentado na Seção 9.

Conclusões, contribuições, ameaças à validade, sugestões de trabalhos futuros e publicações provenientes do estudo são apresentados na Seção 10.

2 METODOLOGIA DE DESENVOLVIMENTO DA PESQUISA

O método científico de uma pesquisa corresponde a um caminho para chegar ao fim de determinado trabalho (GIL, 2012). Há várias maneiras para classificar uma pesquisa sendo os pontos mais tradicionais (SILVA; MENEZES, 2000): i) a natureza da pesquisa; ii) a forma de abordagem do problema; iii) os seus objetivos; e iv) os procedimentos técnicos.

2.1 Classificação da pesquisa

Em relação à natureza, a pesquisa pode ser classificada como **aplicada** ou **tecnológica** (GIL, 2012; JUNG, 2009). Na pesquisa aplicada, o objetivo é gerar conhecimentos para aplicação prática, os quais sejam dirigidos à solução de problemas específicos para a obtenção de um novo produto ou processo (GIL, 2012; JUNG, 2009). Propõe-se a realização de pesquisa com a abordagem **quantitativa** considerando a população de sistemas de software desenvolvidos na linguagem Java e com a tecnologia OO em repositórios de acesso aberto. A pesquisa quantitativa utiliza a elaboração de enunciados analíticos, descrições matemáticas das variáveis e as relações dessas variáveis (GIL, 2012; JUNG, 2009; SAMPIERI; COLLADO; LUCIO, 2013).

Em relação aos objetivos, a pesquisa pode ser classificada como **descritiva**, que visa à identificação, ao registro e à análise de características, de fatores ou de variáveis que se relacionam com o fenômeno ou processo (GIL, 2012; JUNG, 2009). Com estudos descritivos são medidos, avaliados ou coletados dados sobre diversos aspectos ou dimensões relacionadas ao fenômeno ou processo estudado (SAMPIERI; COLLADO; LUCIO, 2013). Do ponto de vista dos procedimentos técnicos, a pesquisa pode ser classificada como **experimental** (GIL, 2012; JUNG, 2009), pois determina se as características

descritas e correlacionadas são capazes de influenciar na qualidade interna de sistemas de software. O objetivo é estudar as causas que determinam tais comportamentos e fenômenos no código estudado e o modo como são produzidas. A análise dos resultados desse estudo foi justificada por meio de técnicas de estatística descritiva e análise multivariada.

A metodologia de desenvolvimento da pesquisa pode ser mais bem esclarecida no desenho de pesquisa. Na subseção 2.2, as fases do desenvolvimento do projeto, as técnicas utilizadas e os instrumentos de coleta e de análise dos dados são apresentados.

2.2 Desenho da pesquisa

O desenho da pesquisa contempla os seus componentes em sequência lógica (YIN, 2010), cuja representação gráfica é apresentada na Figura 1. Como pode ser observado, a pesquisa foi realizada em três etapas:

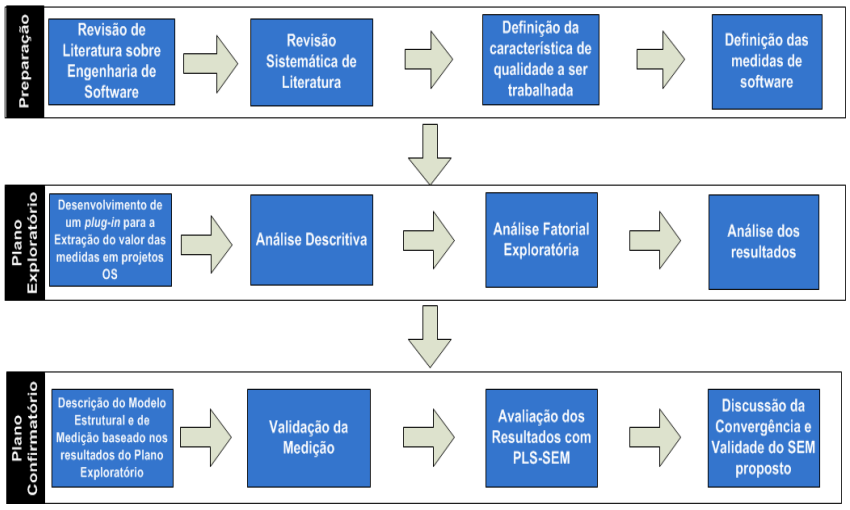


Figura 1 Desenho da metodologia de desenvolvimento da pesquisa

a) **Preparação**. Essa etapa foi destinada ao levantamento bibliográfico e à preparação do *dataset* inicial da pesquisa, com quatro atividades:
   - **Revisão de Literatura sobre Engenharia de Software**. É realizado um levantamento bibliográfico, na internet e em bibliotecas digitais e impressas, de artigos científicos relacionados ao tema, tais como, qualidade de software, medição e técnicas estatísticas em Engenharia de Software;
   - **Revisão Sistemática de Literatura.** Após a revisão de literatura tradicional, para a busca e a compreensão das questões a serem investigadas, foi executada uma Revisão Sistemática de Literatura (RSL). A RSL é uma técnica que visa a identificação, a avaliação e a interpretação de estudos ou pesquisas disponíveis e relevantes a uma questão de pesquisa específica, área temática ou fenômeno de interesse. Os estudos individuais empíricos diretamente relacionados à questão de pesquisa são chamados estudos primários. Estudos secundários visam a revisão, a investigação e a sintetização das evidências identificadas nos estudos primários (KITCHENHAM, 2004). Em seguida, a partir dos resultados da RSL foram definidas as medidas de software utilizadas no estudo e as propriedades da tecnologia de OO que representam a característica Manutenibilidade;
   - **Definição da Característica de Qualidade a ser Trabalhada.** A partir dos resultados da RSL foi identificada a necessidade de avaliar a qualidade interna dos sistemas sob a característica Manutenibilidade, considerando as diferentes medidas, suas propriedades e relevância (medidas consideradas relevantes para a caracterização da qualidade interna de sistemas de software OO por pesquisadores);

- **Definição das Medidas de Software**. Após definida a característica Manutenibilidade foram definidas as medidas de software utilizadas no estudo, juntamente com as propriedades da OO a serem analisadas. A escolha das medidas seguiu recomendações dos resultados obtidos na RSL;

b) **Plano Exploratório**. Essa etapa teve como foco a exploração dos dados, no descobrimento de correlações e de similaridades entre as variáveis, com quatro atividades:
   - **Desenvolvimento de um Plug-in para a Extração do Valor das Medidas em Projetos OS.** Esse plug-in extrai o valor das medidas selecionadas (na etapa Preparação) dos sistemas de software OO OS desenvolvidos em Java, coletados nos repositórios Github e Sourceforge;
   - **Análise Descritiva**. Dados descritivos da amostra e das observações feitas dos domínios são apresentados, por exemplo, a frequência, a média, a variância e o desvio padrão;
   - **Análise Fatorial Exploratória**. A Análise Fatorial Exploratória (AFE) é uma técnica de análise exploratória de dados, cujo objetivo é a análise da estrutura de um conjunto de variáveis interrelacionadas, tornando-se possível a construção de uma escala de medidas para fatores, que podem diretamente, ou não, controlar as variáveis da pesquisa (HAIR et al., 2009; MAROCO, 2010). A finalidade de sua utilização é (i) reduzir dados ou simplificar sua estrutura; (ii) classificar e agrupar; (iii) investigar dependência entre variáveis e; (iv) predizer, elaborar e testar hipóteses (JOHNSON; WICHERN, 2007). No entanto, a AFE foi utilizada com o objetivo de identificar fatores que

caracterizam a qualidade interna de sistemas de software OO, quanto à sua Manutenibilidade. Esses fatores foram formados por medidas que possuem correlação e explicam um comportamento comum para obtenção da qualidade interna desses sistemas;

- **Análise dos Resultados**. Com a utilização da AFE, é discutido o relacionamento entre as variáveis estudadas. A execução e a análise dos resultados da AFE, foram realizadas no software de análise estatística SPSS (Statistical Package for the Social Sciences), na versão 20, para o sistema operacional Windows 7 Ultimate Edition.

c) **Plano Confirmatório**. Essa etapa teve como foco o teste de confirmação do modelo multivariado, extraído da etapa exploratória, e possui quatro atividades:

- **Descrição do Modelo Estrutural e de Medição Baseado nos Resultados do Plano Exploratório**. Foram elaboradas hipóteses com base nos interrelacionamentos descobertos para verificar e avaliar os construtos encontrados na fase Plano Exploratório. Para tanto, foi necessário modelar estatisticamente essas hipóteses, por meio da técnica Modelagem de Equações Estruturais (Structural Equation Modeling - SEM). O modelo buscou caracterizar a qualidade interna de software OO, sob a manutenibilidade preventiva;
- **Validação da Medição**. Os testes de validade dos modelos de medição como o bootstraping e os testes de convergência dos relacionamentos entre os fatores foram realizados. O objetivo foi identificar a presença/ausência de significância das variáveis

no modelo e verificar se havia suporte empírico para o modelo teórico desenvolvido;

- **Avaliação dos Resultados com PLS-SEM**. Após a etapa Validação da Medição, as cargas dos relacionamentos entre os indicadores e os construtos foram analisadas e reportadas. O mapeamento das características de cada construto e do seu efeito no modelo proposto foi elaborado;
- **Discussão da Convergência e Validade do SEM Proposto**. Depois de reportados os resultados das cargas, os efeitos do relacionamento entre os construtos foram reportados. Além disso, a significância do poder de predição dos indicadores e dos construtos para a garantia da qualidade interna dos sistemas de software OO foi discutida.

2.3 Abrangência da pesquisa

A abrangência ou delimitação da pesquisa estabelece limites de investigação (MARCONI; LAKATOS, 2010): a população e a amostragem. Neste trabalho, o tipo de amostragem é estratificada com partilha proporcional ou estratificada proporcional, em que a população estudada encontra-se dividida em estratos (ou camadas, faixas, intervalos, etc). Esse tipo de abordagem consiste na divisão da população (1.037 sistemas de software coletados nos repositórios Github e Sourceforge), em subgrupos homogêneos (domínios de software) (HAIR et al., 2009; MAROCO, 2010).

Para a execução de análises de caráter confirmatório sugere-se amostras maiores, pois são mais estáveis para possíveis replicações em estudos (HAIR et al., 2009). Na literatura, são recomendadas amostras de 300 a 500 indivíduos ou até mais (para casos em que dados perdidos podem exceder 10% da amostra)

(HAIR et al., 2009). Dessa forma, o tamanho de amostra foi definido em 500 sistemas de software (50 sistemas por domínio), pois se encontra no intervalo recomendado para análise.

A amostra final foi constituída por uma amostragem simples dos elementos pertencentes em cada um dos subgrupos, garantida a representatividade dos grupos existentes na população teórica. Esses elementos são obtidos de forma aleatória, pois a probabilidade de cada elemento da população fazer parte da amostra é igual para todos os elementos e todas as amostras são igualmente prováveis (HAIR et al., 2009; MAROCO, 2010). Dessa forma, para a construção do *dataset* da pesquisa, apenas os sistemas de software compiláveis e que atendiam os critérios de seleção foram considerados.

3 QUALIDADE DE SOFTWARE

3.1 Considerações iniciais

Medir atributos ou propriedades de projetos, de processos e de sistemas de software pode ajudar as empresas a alcançarem seu principal objetivo: aumentar seus lucros com a comercialização de produtos e de serviços, com custos mais baixos, a qualidade mais alta e menor tempo de entrada no mercado. Para avaliar a qualidade de sistemas de software de forma geral, organizações nacionais e internacionais vêm publicando e revisando normas que orientam, por exemplo, o desenvolvimento, a aquisição, a manutenção e a avaliação de sistemas de software.

Por meio da análise automatizada de software utilizando medidas de software, podem ser identificados importantes indicadores de fatores de qualidade, que podem ser utilizados para prever características em um software (ANDA, 2007; KAYARVIZHY; KANMANI, 2011). Nesta seção, os tópicos relacionados à qualidade de sistemas de software, por exemplo, os conceitos e a fundamentação, a sua importância para os envolvidos no processo de desenvolvimento de software e como a qualidade pode ser avaliada e caracterizada são abordadas.

Esta seção está organizada da seguinte forma. Conceitos de qualidade são brevemente apresentados na subseção 3.2. Importância da qualidade para o desenvolvimento de software é abordada na subseção 3.3. Tipos de qualidade de software, como a qualidade de projeto, de processo e de produto, são tratados na subseção 3.4. A característica Manutenibilidade, o foco do estudo, é discutida na subseção 3.5. Processo de avaliação da qualidade interna de software e os atributos utilizados como referência no estudo são descritos na subseção 3.6.

3.2 Conceitos

Embora a qualidade seja um fator crítico para o sucesso de um software, esse conceito é complexo de definir, de descrever, de compreender e de mensurar (KITCHENHAM; WALKER, 1989). Em primeiro lugar, qualidade não é uma ideia única, mas um conceito multidimensional. As dimensões da qualidade incluem interesses e pontos de vista sobre essa entidade, além dos atributos de qualidade. Em segundo lugar, para qualquer conceito, há níveis de abstração; quando se referem à qualidade, seu significado pode ser mais amplo ou específico. Por fim, o termo "qualidade" faz parte da linguagem cotidiana e os usos populares e profissionais do seu significado podem ser diferentes (KAN, 2002).

O conceito de qualidade deve ser descrito em uma definição viável. Inicialmente, qualidade foi definida como adequação ao uso (JURAN; GRYNA JÚNIOR, 1970); em seguida, como conformidade com os requisitos (CROSBY *apud* PRESSMAN, 2014). As duas definições de qualidade são essencialmente similares, cuja diferença é o conceito de adequação ao uso implicar um papel mais significativo às necessidades e às expectativas dos *stakeholders*. Contudo, elas têm sido adotadas e utilizadas por profissionais e, quando ligadas a software, tornaram-se mais robustas, conforme padronizadas nas normas criadas pela *International Organization for Standardization* (ISO) e por outras entidades vinculadas à garantia da qualidade de software, como o *Institute of Electrical and Electronics Engineers* (IEEE) e o *Software Engineering Institute* (SEI).

Qualidade de um sistema de software é o grau em que as características de um produto/serviço satisfaz as necessidades explícitas e implícitas dos *stakeholders* agregando valor a esse produto/serviço (INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 2011). O IEEE apresenta uma definição similar de qualidade à ISO/IEC 25000, que consiste no grau em que

um sistema, um componente ou um processo atende os requisitos especificados e as necessidades ou expectativas do cliente/usuário (SOFTWARE ENGINEERING STANDARDS COMMITTEE OF THE IEEE COMPUTER SOCIETY, 2006).

Dada à complexidade desse conceito, a qualidade de um sistema de software pode ser definida pela noção de que suas várias partes devem atender especificações predeterminadas (CROSBY, 1979 *apud* PRESSMAN, 2014). Essas especificações de qualidade são elencadas por meio de normas e de modelos, por exemplo, ISO/IEC 25000, CMMI e MPS.BR.

3.3 Importância

A qualidade é um fator importante em qualquer setor da vida humana. A qualidade de um produto/serviço está intimamente relacionada com a amplitude que ele pode oferecer ao usuário. Com o mundo cada vez mais globalizado, organizações crescentemente buscam a qualidade como forma de sobressair perante a concorrência e, em muitos casos, um pré-requisito para continuarem funcionando (HAMID; HASAN, 2010; KAN, 2002; SINGH; KANNOJIA, 2013).

Em organizações desenvolvedoras de sistemas de software há constantemente a necessidade de alterar esses sistemas, pois seus requisitos evoluem e as necessidades de seus usuários mudam (HAMID; HASAN, 2010; SINGH; KANNOJIA, 2013). No entanto, essas alterações são inevitáveis (LEHMAN; BELADY, 1985), tendo em vista que os sistemas, para continuarem úteis às organizações devem atender os requisitos de seus usuários. Por outro lado, essas alterações podem deteriorar a estrutura interna dos sistemas tornando-os difíceis de serem analisados, entendidos, compreendidos, alterados e testados (ABUASAD; ALSMADI, 2012).

Saber se a qualidade de um sistema de software foi adequadamente projetada ou se ela foi afetada de sobremaneira, após manutenções, não é trivial (ABUASAD; ALSMADI, 2012; PING, 2010). Dessa forma, a facilidade em realizar manutenção nesses sistemas é fator crítico para continuarem a ser utilizados pelos seus usuários (INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 2011; PING, 2010; SUNDAY, 1989). Na área Engenharia da Qualidade de Software, a principal meta é a construção de sistema de software de alta qualidade (RUNESON; ISACSSON, 1998). Essa qualidade pode ser conseguida definindo modelos de procedimentos e padrões que descrevem boas práticas a serem seguidas pelos envolvidos no processo de desenvolvimento do sistema de software (PRESSMAN, 2014; SOMMERVILLE, 2015).

A utilização de indicadores de qualidade de software é benéfica para engenheiros e gerentes de software determinarem confiabilidade do sistema, estimarem e priorizarem os itens de trabalho, com foco em áreas que necessitam de testes, de inspeções e, em geral, de identificação de pontos críticos para gerir em situações imprevistas (HAMID; HASAN, 2010; KAN, 2002; SINGH; KANNOJIA, 2013). Os *stakeholders* utilizam as avaliações de qualidade de sistemas de software, como base para importantes decisões incluindo a melhoria da qualidade do produto, as realizações de aquisições de grande porte e o acompanhamento dos contratos (JUNG; KIM; CHUNG, 2004). Ainda que os componentes sejam de boa qualidade, se os processos forem inadequados, o produto final terá baixa qualidade. Para garantir a qualidade dos produtos/serviços é necessário garantir a melhoria contínua dos processos que os geraram (INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 2004).

A garantia da qualidade envolve um conjunto de atividades voltadas para avaliar o processo pelo qual os produtos são desenvolvidos ou

manufaturados, visando fornecer confiança necessária de que estejam em conformidade com os requisitos técnicos especificados (INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 2011; RUNESON; ISACSSON, 1998; SOFTWARE ENGINEERING STANDARDS COMMITTEE OF THE IEEE COMPUTER SOCIETY, 2006). Essas atividades incluem a elaboração de um plano de gerenciamento da qualidade, a definição das medidas de qualidade a serem utilizadas na avaliação, a elaboração do plano de melhoria de processos, o plano de solicitação de mudanças e a definição das ações corretivas e preventivas a serem desenvolvidas (INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 2004; SOFTWARE ENGINEERING STANDARDS COMMITTEE OF THE IEEE COMPUTER SOCIETY, 2006).

Com a influência direta das atividades de garantia da qualidade no sucesso de um sistema de software, ter ciência das características desses sistemas torna-se importante passo para identificar componentes com muita ou pouca qualidade (SOMMERVILLE, 2015). Nesse sentido, a medição da qualidade do sistema de software durante e depois do processo de desenvolvimento pode ser uma atividade efetiva para a construção de modelos que descrevem essas características estruturais de sistemas de software (KAYARVIZHY; KANMANI, 2011). Para uma organização ser reconhecida como de qualidade é fundamental que os colaboradores sintam-se responsáveis e comprometidos com o resultado final, bem como saberem, ao longo do processo de produção, exatamente o que é exigido como satisfatório para cada atividade, seja na qualidade do projeto, do processo, ou do produto (CAMPOS, 1992).

3.4 Tipos

A qualidade de software pode ser definida e alcançada sob três aspectos:

a) **Qualidade do Projeto**. É definida como o grau até o qual um conjunto de características inerentes satisfaz as necessidades (PMBOK GUIDE, 2013). Um projeto com qualidade é aquele concluído em conformidade com os requisitos, as especificações estabelecidas pelos *stakeholders* envolvidos e a adequação ao uso (PMBOK GUIDE, 2013). Dentre as abordagens utilizadas para garantir a qualidade de um projeto estão: i) GQM (*Goal-Question-Metric*), que se baseia no pressuposto de, para mensurar qualidade, uma organização especificar as metas para si mesma, para seus projetos e traçar os objetivos para alcançar essas metas (VAN SOLINGEN; VAN BERGHOUT, 1999) e; ii) *Six Sigma*, que visa melhorar a lucratividade de qualquer empresa, aumentar a participação de mercado, reduzir custos e aperfeiçoar as operações da empresa que o utiliza (BREYFOGLE; CUPELLO; MEADOWS, 2001);

b) **Qualidade do Processo**. Refere-se ao grau em que o processo garante a qualidade de um produto (INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 2004). A suposição fundamental da gerência de qualidade é a qualidade do processo de desenvolvimento afetar diretamente a qualidade dos produtos entregues (SOMMERVILLE, 2015). O processo pelo qual os produtos são desenvolvidos é avaliado por meio de atividades da Garantia da Qualidade (SOFTWARE ENGINEERING STANDARDS COMMITTEE OF THE IEEE COMPUTER SOCIETY, 2006). A qualidade do processo é tratada em normas internacionais, tais como, ISO/IEC 15504 - “Tecnologia da Informação - Avaliação de Processo” (INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 2004) e ISO/IEC

12207 - “Engenharia de Sistemas e Software - Processos de Ciclo de Vida de Software” (INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 2008), e em modelos de qualidade de software, por exemplo, o *Capability Maturity Model Integration* - CMMI (SOFTWARE ENGINEERING INSTITUTE, 2010) e o modelo de Melhoria de Processo de Software e Serviços Brasileiro - MPS.BR (SOFTEX, 2013);

c) **Qualidade do Produto**. Refere-se aos atributos de qualidade interna e externa. Os atributos de qualidade externa garantem que o produto está em conformidade com as necessidades do usuário. Os atributos de qualidade interna correspondem à qualidade da organização do código ou de sua complexidade (INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 2011). Esses atributos diferenciam-se pela forma como são aferidos e, em conjunto, compõem a qualidade do software em si. Eles podem ser definidos a partir de oito características de modo a agregar valor aos *stakeholders* (INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 2011). A norma com o objetivo de apoiar a avaliação da qualidade de sistemas de software é a ISO/IEC 25000 - “Engenharia de Software - Requisitos e Avaliação da Qualidade de Produtos de Software”, que substituiu as normas ISO/IEC 9126 e ISO/IEC 14598. A norma ISO/IEC 25051 - “Requisitos de Qualidade de Produto de Software Comercial de Prateleira e Instruções para Teste” substituiu a norma ISO/IEC 12119 - “Tecnologia de Informação - Pacotes de Software - Teste e Requisitos de Qualidade”. Essas normas definem a forma em que características e atributos de um sistema de software devem ser

avaliados e sugerem medidas que podem ser utilizadas para quantificar a qualidade desses sistemas (PING, 2010).

Normas e modelos de qualidade são utilizados por profissionais da área Engenharia de Software como referências para especificação e para avaliação da qualidade de software sob a perspectiva dos três tipos de qualidade. Especificamente, na norma ISO/IEC 25000, estão descritos dois modelos de qualidade de sistemas de software, apresentando características e subcaracterísticas (INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 2005; INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 2011):

a) **Qualidade do produto**: composto de oito características (subdivididas em subcaracterísticas) que se relacionam com as propriedades estáticas de software e as propriedades dinâmicas do sistema de computador;
b) **Qualidade em uso**: composto por cinco características (mas, possuem subcaracterísticas) que se relacionam com o resultado da interação, quando um produto é utilizado em um contexto particular de uso.

O modelo do ciclo de vida da qualidade de software aborda a qualidade do sistema de software em três principais fases do ciclo de vida (Figura 2) (INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 2011):

a) **Produto em desenvolvimento**: é o tema da qualidade interna do software;
b) **Produto em fase operacional**: é o tema da qualidade externa do software;
c) **Produto a ser utilizado**: é o tema da qualidade do uso.

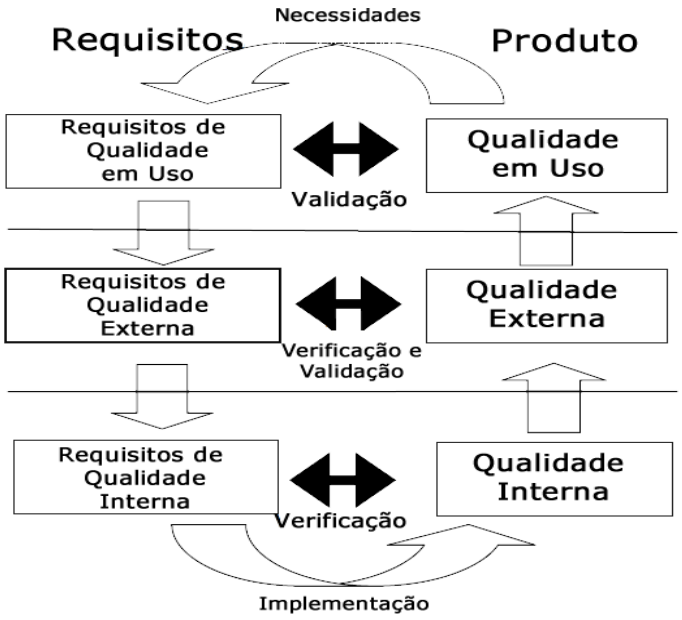


Figura 2 Modelo do ciclo de vida da qualidade de sistemas de software
Fonte: Adaptado de International Organization for Standardization (2011).

O modelo do ciclo de vida da qualidade de sistemas de software também indica que a implementação dessa qualidade requer um processo semelhante ao processo de desenvolvimento de software, para cada tipo de qualidade: i) requisitos; ii) implementação; e iii) validação dos resultados.

Os **requisitos de qualidade em uso** especificam o nível exigido de qualidade do ponto de vista do usuário final. Esses requisitos devem ser especificados por meio das medidas, para serem utilizados como critérios

quando um sistema for validado. Assim, para obter um sistema que satisfaça as necessidades do usuário, normalmente requer que seja utilizada uma abordagem iterativa para o desenvolvimento de software e que tenha *feedback* contínuo do produto final sob a perspectiva do usuário (INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 2011).

Os **requisitos de qualidade externa** de software especificam o nível de qualidade requerida sob o ponto de vista externo. Eles incluem requisitos derivados das necessidades de qualidade do usuário, incluindo os **requisitos de qualidade em uso**. Esses requisitos são utilizados como meta para validação em vários estágios de desenvolvimento. Eles devem ser abrangidos pelas características de qualidade definidas, sendo declarados na especificação de requisitos de qualidade, usando medidas externas. Além disso, convém que eles sejam traduzidos em **requisitos de qualidade interna** e usados como critérios na avaliação do sistema de software (INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 2011).

Os **requisitos de qualidade interna** de software especificam o nível de qualidade exigido a partir da visão interna do produto. Esses requisitos são utilizados para especificar as propriedades dos produtos intermediários e podem incluir modelos estáticos e dinâmicos, outros documentos e código. Além disso, podem ser usados como metas para validação em vários estágios de desenvolvimento. Eles também podem ser utilizados para definir estratégias de desenvolvimento e critérios de avaliação e de verificação durante o desenvolvimento. Isso pode incluir a utilização de medidas adicionais (por exemplo, reusabilidade), as quais estão fora do escopo da norma ISO/IEC 25010. Convém que **requisitos de qualidade interna** sejam definidos quantitativamente, utilizando medidas internas (INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 2011).

3.5 Manutenibilidade

Na Engenahria de Software, Manutenibilidade é uma característica da qualidade de software que se refere à facilidade de um software de ser modificado, a fim de corrigir defeitos, adequar-se a novos requisitos, aumentar a suportabilidade ou adequar-se a um ambiente novo (INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 2011). Na norma ISO-25010, a característica Manutenibilidade pertence ao modelo de qualidade do produto e possui cinco subcaracterísticas (Figura 3):

a) **Modularidade**: grau em que um sistema de software é composto de componentes discretos, tal qual a mudança de um componente deve ter, ou seja, o menor impacto possível em outros componentes;

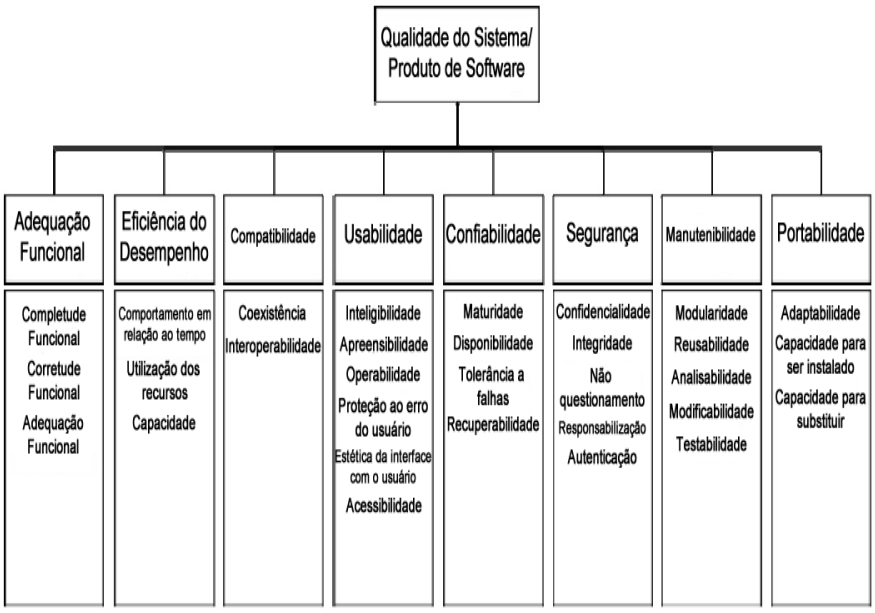


Figura 3 Modelo de qualidade de sistemas de software
Fonte: Adaptado de International Organization for Standardization (2011).

b) **Reusabilidade**: grau em que um componente pode ser utilizado em mais de um sistema ou na construção de outros componentes;
c) **Analisabilidade**: grau de eficácia e de eficiência com que pode ser avaliado o impacto de uma alteração sobre um produto ou sistema, em um ou mais dos seus componentes, ou para diagnosticar um produto para deficiências ou causas de falhas, ou para identificar peças a serem modificadas;
d) **Modificabilidade**: grau para o qual um sistema de software pode ser modificado de forma eficaz e com eficiência, sem a introdução de defeitos ou degradaçãoda qualidade do sistema existente;
e) **Testabilidade**: grau de eficácia e de eficiência com que podem ser estabelecidos critérios de teste para um sistema, produto ou componente, e testes podem ser realizados para determinar se esses critérios foram cumpridos.

Essas subcaracterísticas podem ser medidas por meio de **medidas de qualidade externas** e de **medidas de qualidade internas**. Uma **medida de qualidade externa** é utilizada para medir a qualidade do sistema de software por mensurar o comportamento do sistema do qual faz parte. Essas medidas podem ser utilizadas durante a fase de testes do processo de ciclo de vida e durante quaisquer etapas operacionais (INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 2011). Antes de adquirir ou de utilizar um sistema de software, convém que ele seja avaliado utilizando medidas baseadas nos objetivos de negócio e relacionadas ao uso e na exploração e na gestão do sistema em um ambiente técnico e organizacional específico. As medidas externas oferecem aos usuários, avaliadores, executores de teste e desenvolvedores, os benefícios de poderem avaliar a qualidade do sistema de

software durante seu teste ou operação (INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 2011).

Uma **medida de qualidade interna** pode ser aplicada a um artefato de software não executável, por exemplo, especificação ou código, respectivamente, durante o projeto e a codificação. Convém que, no desenvolvimento de um sistema de software, os produtos intermediários sejam avaliados utilizando medidas internas, que medem propriedades intrínsecas, incluindo aquelas que podem ser derivadas de um comportamento simulado. O propósito básico dessas medidas é assegurar que a qualidade externa e em uso requeridas, sejam alcançadas, exemplos são encontrados na norma ISO/IEC 25023 (INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 2011). Elas oferecem aos usuários, avaliadores, executores de teste e desenvolvedores, os benefícios de poderem avaliar a qualidade de artefatos de software e considerarem questões relativas a qualidade, bem antes do sistema de software tornar-se executável (INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 2011).

Portanto, as medidas internas medem atributos internos pela análise das propriedades estáticas dos artefatos de software intermediários ou preparados para entrega. Da mesma forma, as medidas externas podem ser indicadoras de atributos externos. As medições internas utilizam a quantidade ou a frequência de elementos que compõem o software e que aparecem, por exemplo, em declarações de código, no gráfico de controle e nas representações de fluxo de dados e de transição de estados. Podem ser avaliadas por meio de revisão, de inspeção, de simulação e/ou de ferramentas automatizadas (INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 2011).

3.6 Avaliação da qualidade interna de software

Para a qualidade interna de um software ser avaliada, na especificação ou no código, é necessário utilizar as medidas de software (INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 2011). Porém, as medidas tradicionais não são suficientes para caracterizar sistemas de software OO, pois possuem características peculiares à tecnologia de desenvolvimento (TIAN; CHEN; ZHANG, 2008; HAMID; HASAN, 2010).

Nesta subseção, são apresentadas medidas utilizadas no estudo da qualidade interna dos sistemas de software selecionados. Além disso, as propriedades ou as características pelas quais as medidas são especializadas (e.g., complexidade, acoplamento e coesão) e os conceitos inerentes à tecnologia OO (e.g., herança e polimorfismo) são abordados.

3.6.1 Medidas de software

Medir atributos e propriedades de um sistema de software é importante aspecto na determinação da qualidade desse sistema. A medição é uma importante ferramenta de apoio à melhoria de projetos, de processos, de produtos e à gestão das atividades do ciclo de vida de software (PING, 2010). Desse modo, medidas de software apoiam a avaliação da viabilidade de planos de projeto, bem como os custos, a qualidade e a capacidade de manutenibilidade do sistema e é útil no monitoramento do cumprimento das atividades do projeto em relação ao planejamento (BASILI, 1985).

Em 2002, a ISO publicou a norma internacional ISO/IEC 15939, que contém definições dos termos utilizados no processo de medição de software, incluindo o termo “medida” (*measure*) no lugar do termo “métrica” (*metric*) (AL-QUTAISH, 2009). Uma medida é um valor atribuído a uma entidade para

caracterizar um atributo ou uma propriedade cuja faixa de valores, o domínio ou o tipo, devem ser especificados (INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 2011; KAN, 2002). As medidas podem ser aplicadas a projetos (KAN, 2002), a processos (INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 2010; KAN, 2002) e a produtos (INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 2010; KAN, 2002). Diversas medidas de sistemas de software foram e são propostas por pesquisadores, pela indústria e por grupos de trabalhos relacionados a organizações nacionais e internacionais, como as medidas propostas na norma ISO/IEC 25000. Apesar de estarem classificadas em diferentes tipos, as medidas devem ter características em comum (KOSCIANSKI; SOARES, 2006):

a) Devem ter **significância**, pois seus resultados devem agregar informação útil à avaliação da qualidade interna do sistema de software avaliado;
b) Devem ter **custo** e **complexidade** de aplicação compatível a avaliação, levando a economia de recursos;
c) Devem ser **repetíveis** e **reproduzíveis**, significando que os resultados de uma medição em ambientes semelhantes devem ser os mesmos;
d) Devem ser **objetivas** e **imparciais**, vinculadas a técnicas matemáticas e estatísticas, pois devem prover evidências para sua validação.

Além disso, podem ser coletadas, ao longo do processo de desenvolvimento de software para auxiliar no monitoramento e no controle da qualidade de sistemas (TIAN; CHEN; ZHANG, 2008), bem como após o sistema ter sido desenvolvido (MUTHANNA et al., 2000). Essas medidas

podem ser de tipos diferentes e são utilizadas para caracterizar quantitativamente algumas propriedades de projetos, de processos e de produtos, tais como, defeitos de software, testes, custo e esforço de manutenção, bem como características específicas de uma tecnologia como herança na OO (PRASAD; NAGAR, 2009).

Ao final da década de 80, vários estudos concluíram que medidas de software tradicionais não eram suficientes para analisar e caracterizar a qualidade de sistemas de software OO (HAMID; HASAN, 2010). Novas medidas foram propostas na literatura para avaliar a qualidade estrutural de código OO (BIEMAN; KANG, 1995; BRIAND; DEVANBU; MELO, 1997; CHIDAMBER; KEMERER, 1991; CHIDAMBER; KEMERER, 1994; HENDERSON-SELLERS, 1996; HITZ; MONTAZERI, 1995; LAKE; COOK, 1994; LEE et al., 1995; LI; HENRY, 1993; LORENZ; KIDD, 1994; MARTIN; MARTIN, 2011; TEGARDEN; SHEETZ; MONARCHI, 1992). Tais medidas destinam-se a proporcionar uma forma de avaliar a qualidade interna de um sistema de software.

O primeiro conjunto de medidas para OO foi proposto por Chidamber e Kemerer (CK), conhecidas como medidas MOOSE (*Metrics for Object-Oriented Software Engineering*) (CHIDAMBER; KEMERER, 1991; CHIDAMBER; KEMERER, 1994). Essas medidas podem auxiliar os usuários no entendimento da complexidade do sistema de software, na detecção de falhas, na previsão de certos resultados e na qualidade externa de sistemas de software, tais como, defeitos, testes e esforço de manutenção (PRASAD; NAGAR, 2009). As medidas que compõem a suíte CK são (CHIDAMBER; KEMERER, 1991; CHIDAMBER; KEMERER, 1994; KULKARNI; KALSHETTY; ARDE, 2010):

a) ***Weighted Methods in a Class*** **(WMC)**: corresponde a soma da complexidade dos métodos de uma classe. Quando classes

apresentam valor alto para a medida WMC, significa que tendem a ser específicas, destinando-se às necessidades individuais, o que restringe sua reutilização. Indica o quanto de esforço é necessário para desenvolver e manter uma determinada classe;

b) ***Depth of the Inheritance Tree*** **(DIT)**: corresponde à quantidade máxima de superclasses acima da classe em questão. Valor alto para a medida DIT pode indicar que a classe em questão herda: i) muitas características comuns de outras classes, sugerindo que suas superclasses estão preparadas para reutilização, visto que provavelmente possuem alto nível de abstração; e ii) muitos serviços, o que pode aumentar sua complexidade;

c) ***Number of Children*** **(NOC)**: corresponde à quantidade de subclasses imediatas subordinadas a uma classe na hierarquia de classes. Valor alto para a medida NOC indica que uma superclasse possui muitos filhos que necessitaram desenvolver características próprias, representando baixo nível de abstração, pois podem existir poucas características em comum entre as classes filhas;

d) ***Coupling Between Objects*** **(CBO)**: corresponde à quantidade de outras classes cujos métodos ou atributos são usados por uma classe. Valor alto para a medida CBO indica que a classe possui muitos relacionamentos, o que dificulta sua reutilização e aumenta sua complexidade, visto que a classe se torna dependente de outras para efetuar suas operações;

e) ***Lack of COhesion in Methods*** **(LCOM)**: corresponde a quantidade de acessos a um ou mais atributos em comum, pelos métodos da própria classe. É a contagem da quantidade de similaridade de pares de métodos iguais a zero, menos a contagem de pares de métodos

cuja semelhança não é zero. Valor alto para a medida LCOM indica que a classe é menos coesa;

f) ***Response For a Class*** **(RFC)**: corresponde à capacidade de resposta que a classe tem ao receber mensagens de seus objetos. É a quantidade de métodos locais e a quantidade de métodos que podem ser chamados em resposta a uma mensagem recebida por um objeto ou por algum método da classe. Valor alto para a medida RFC indica que maior é a possibilidade da classe em atender as mensagens recebidas.

No entanto, a suíte CK não aborda alguns aspectos de sistemas de software OO. Estudos subsequentes complementaram a pesquisa, como a suíte de medidas de Li e Henry. Nesse conjunto, são propostas cinco medidas (LI; HENRY, 1993):

a) ***Data Abstraction Coupling*** **(DAC)**: corresponde à quantidade de tipos abstratos de dados definidos em uma classe e dependentes das definições de outras classes. Quanto mais tipos abstratos de dados uma classe tiver, maior é seu acoplamento com outras classes;
b) ***Message Passing Coupling*** **(MPC)**: corresponde à quantidade de indicações de envio definidas em uma classe. É utilizada para medir a complexidade da passagem de mensagens entre classes. Uma vez que o padrão da mensagem é definido por uma classe e utilizado pelos seus objetos, a medida indica quantas mensagens são transmitidas entre os objetos das classes;
c) ***Number of Methods*** **(NOM)**: corresponde à quantidade de métodos locais definidos em uma classe. Essa quantidade pode indicar a

propriedade de operação de uma classe. Quanto mais métodos ela tem, mais complexa ela se torna;

d) ***SIZE1***: corresponde à quantidade de pontos e vírgulas em uma classe;
e) ***SIZE2***: corresponde à quantidade de propriedades de uma classe, cujo cálculo é a soma da quantidade de atributos e de métodos locais de uma classe.

Na suíte de medidas de Bieman e Kang (BIEMAN; KANG, 1995), foram definidas duas medidas de coesão baseadas nas conexões diretas e indiretas dos pares de métodos:

a) ***Tight Class Cohesion*** **(TCC)**: corresponde à quantidade de métodos diretamente conectados em uma classe;
b) ***Loose Class Cohesion*** **(LCC)**: corresponde à quantidade de métodos indiretamente conectados em uma classe.

Alguns pesquisadores propuseram melhorias em medidas estabelecidas (CHIDAMBER; KEMERER, 1994; HENDERSON-SELLERS, 1996; HITZ; MONTAZERI, 1995; LI; HENRY, 1993;). Por exemplo, a medida LCOM, também conhecida como LCOM1, definida inicialmente por CK, ganhou quatro variações (CHIDAMBER; KEMERER, 1991):

a) A medida **LCOM2** calcula o percentual entre os métodos que não acessam um atributo específico e a média dos atributos na classe. Se a quantidade de métodos ou de atributos é zero, o valor da medida **LCOM2** é indefinido e apresentado como zero (CHIDAMBER; KEMERER, 1994; HENDERSON-SELLERS, 1996);

b) Utilizando um grafo não direcionado, em que um nó representa cada método que partilha, de pelo menos uma variável de instância, foi definida uma nova versão para LCOM - **LCOM3**, cujo cálculo é a quantidade de componentes conectados em um grafo (LI; HENRY, 1993);
c) Uma extensão de LCOM3 foi proposta (HITZ; MONTAZERI, 1995) - **LCOM4** -, caracterizando a adição de uma aresta entre um par de métodos, representando um método que invoca o outro (avaliação da conectividade dos componentes quando o grafo tem um componente);
d) Outra variação de LCOM foi proposta (HENDERSON-SELLERS, 1996) - **LCOM5** -, que considera a quantidade de métodos que referencia cada atributo.

3.6.2 Propriedades avaliadas

Conceitos de OO tornaram-se bem estabelecidos na Engenharia de Software nas últimas décadas (PRESSMAN, 2014). A OO é uma tecnologia para o desenvolvimento de sistemas de software que organiza o problema e a sua solução como uma coleção de objetos discretos, em que a estrutura de dados e seu comportamento são incluídos nessa representação (PFLEEGER; ATLEE, 2010). Pode-se reconhecer uma representação OO por suas características (PFLEEGER; ATLEE, 2010; PRESSMAN, 2014):

a) **Abstração**. Abstrair a identidade ou os dados organizados em entidades é essencial na construção de sistemas de software OO, pois auxilia na representação dos diferentes pontos de vista a serem incorporados durante o processo de desenvolvimento. Juntamente

com o conceito de abstração há o conceito de estabilidade. Um componente estável também deve ser abstrato para sua estabilidade não impedir que ele seja estendido. Por outro lado, um componente instável deve ser concreto, pois sua instabilidade permite que o código concreto dentro dele seja facilmente alterado. Assim, se o componente precisa ser estável, também deve consistir em classes abstratas para que possa ser estendido. Componentes estáveis e extensíveis são flexíveis e não restringem demasiadamente o projeto (MARTIN; MARTIN, 2011). A ideia é um componente não ser abstrato demasiadamente, nem rígido para dificultar mudanças (MARTIN; MARTIN, 2011);

b) **Encapsulamento**. Para um sistema de software bem arquitetado, é necessário ocultar detalhes de implementação irrelevantes aos usuários. A ideia de separar o sistema em partes, de forma mais isolada possível, é chamada de encapsulamento, cujo objetivo é controlar o acesso aos atributos e aos métodos de uma classe;

c) **Herança**. Em sistemas de software OO, classes podem ser organizadas em hierarquias, que possuem semelhanças ou diferenças entre si. A herança é uma das características da tecnologia OO (SHAHEEN; SHAHEEN, 2008) que consiste em permitir que classes compartilhem atributos e métodos utilizando uma relação de hierarquia (subclasses herdem características de uma superclasse) (PRESSMAN, 2014). Pesquisas (CHIDAMBER; KEMERER, 1994; RAJNISH; BHATTACHERJEE, 2008) indicam que o uso de herança reduz a quantidade de manutenções necessárias em um sistema de software e alivia a carga de teste. Da mesma forma, a reutilização de um sistema utilizando herança tem como resultado a produção de um sistema mais sustentável, compreensível e confiável

(BASILI; BRIAND; MELO, 1996; RAJNISH; BHATTACHERJEE, 2008).

Alguns outros conceitos relacionados às características descritas, utilizados na avaliação da característica manutenibilidade, por meio de atributos internos de sistemas de software OO, são:

a) **Complexidade**. A complexidade é um termo frequentemente aplicado à interação entre a arquitetura interna de um sistema de software, por exemplo, a quantidade de componentes e a quantidade e a natureza das relações entre esses componentes (KEARNEY et al., 1986; CHOWDHURY; ZULKERNINE, 2011). Um sistema com código complexo é difícil de ser entendido, mantido e testado (MCCABE, 1976; SHIN et al., 2011). Consequentemente, esse código pode ser mais vulnerável que código mais simples (SHIN et al., 2011). Estudos empíricos têm mostrado que as medidas de complexidade, como a medida Complexidade Ciclomática (CC) e as medidas derivadas, por exemplo, a medida WMC, podem ser boas preditoras de características de qualidade, como testabilidade e manutenibilidade em sistemas de software, construídos em linguagens de programação tradicionais, imperativas e orientadas a objetos (OLAGUE et al., 2008);
b) **Acomplamento e Coesão**. Esses conceitos correspondem ao relacionamento de dependência entre seus componentes (ALLEN; KHOSHGOFTAAR, 1999). A coesão mede a associação entre dois componentes de um sistema de software em relação à realização de uma dada tarefa (KOSCIANSKI; SOARES, 2006). Pode-se dizer que um componente é coeso se ele representa operações ou funções

> com características semelhantes. O acoplamento mede o grau de associação entre dois componentes (KOSCIANSKI; SOARES, 2006). Uma vez que componentes estejam fortemente acoplados, os erros em um deles podem resultar em erros em outro (OFFUTT; ABDURAZIK; ALEXANDER, 2000). Um exemplo de acoplamento é a utilização de tipos como variáveis globais. De maneira geral, a importância de acoplamento e de coesão como principais atributos relacionados à qualidade interna de sistemas de software é esperada entre engenheiros de software. Para manter esses sistemas de alta qualidade, os desenvolvedores precisam criá-los com uma arquitetura de baixo acoplamento e alta coesão (HUSEIN; OXLEY, 2009).

3.7 Considerações finais

Nesta seção, foi apresentada uma revisão teórica sobre a Qualidade de Software e seus principais aspectos. Foram tratados os fundamentos da qualidade, incluindo a conceituação sobre as características de qualidade que um sistema de software deve apresentar, bem como o conceito de medidas de software, utilizadas para avaliar e quantificar a qualidade de software.

Dessa forma, a fundamentação dos conceitos teóricos de Qualidade de Software em seus três tipos, seus padrões, seus modelos e suas normas mais tradicionais permitem nortear o estudo, para que o entendimento acerca do objetivo do trabalho seja esclarecido. Esse trabalho propõe a tradução dos princípios explicados e um modelo baseado em técnicas estatísticas, com parâmetros definidos em uma amostra de sistemas de software, cujas recomendações os desenvolvedores possam aplicar em seu processo de

desenvolvimento e de manutenção para a melhoria contínua de seus sistemas. Tais técnicas utilizadas no estudo são apresentadas na Seção 4.

4 TÉCNICAS ESTATÍSTICAS EXPLORATÓRIAS E CONFIRMATÓRIAS - CORRELAÇÃO ENTRE MEDIDAS DE QUALIDADE INTERNA DE SOFTWARE

4.1 Considerações iniciais

A atividade responsável por transformar um conjunto de dados em formas objetivas para um pesquisador ser capaz de interpretá-los, consiste na análise de dados (MAROCO, 2010). Diversas são as técnicas e as abordagens utilizadas para analisar dados, sejam qualitativos ou quantitativos. Neste capítulo são discutidas técnicas utilizadas por pesquisadores da área para executar análise quantitiva de dados (neste trabalho, são as medidas de software).

Este capítulo está organizado da seguinte forma. Uma visão geral sobre pesquisa quantitativa é apresentada na subseção 4.2. Conceitos sobre análise multivariada de dados e de técnicas estatísticas relacionadas são explorados na subseção 4.3. Tipos de técnicas multivariadas utilizadas neste trabalho são discutidos na subseção 4.4.

4.2 Pesquisa quantitativa e estatística descritiva - conceitos iniciais

A pesquisa quantitativa pode ser definida como a investigação empírica e sistemática de fenômenos sociais por meio técnicas estatísticas, matemáticas e computacionais (GIVEN, 2008). A análise de dados ocorre a partir da formulação de modelos de descrições intuitivas do pesquisador ou do indivíduo pesquisado (JUNG, 2009). Na pesquisa quantitativa, a análise obedece um plano preestabelecido, confirmando as hipóteses da pesquisa ou descobertas por dedução, utilizando dados que representam uma população específica (amostra), a partir da qual, os resultados são generalizados (APPOLINÁRIO, 2004).

No processo da análise estatística, o pesquisador depara-se com “algo” que precisa ser medido, controlado ou manipulado durante o processo de investigação. Esse “algo” designa-se por variável (MAROCO, 2010). Dessa forma, os objetos de um estudo estatístico são as variáveis e as informações que podem ser obtidas. Quanto à abrangência ou à delimitação da pesquisa, dois outros conceitos estabelecem os limites da investigação (MARCONI; LAKATOS, 2010): i) a população; e ii) a amostragem.

Em terminologia estatística, uma população é um grupo de objetos, de eventos, de observações ou de outras “coisas” que podem ser agregáveis e sobre a(s) qual(is) o pesquisador deseja generalizar (HAIR et al., 2009; MARCONI; LAKATOS, 2010; MAROCO, 2010). Por exemplo, com a investigação sobre a qualidade interna de software em grande quantidade de sistemas de software *open-source*, o objetivo foi obter resultados que pudessem ser generalizáveis para esse tipo de sistema. Definida a população em estudo, o próximo passo é definir como selecionar os objetos para compor a amostra (HAIR et al., 2009; MARCONI; LAKATOS, 2010; MAROCO, 2010). Com a amostra devidamente construída e classificada, torna-se possível a caracterização dos elementos da pesquisa por meio da estatística descritiva e, consequentemente, com uso de técnicas e de procedimentos que permitem proporcionar ao pesquisador, um grau de confiabilidade e de generalização em suas afirmações, para a população teórica, por meio da inferência estatística (HAIR et al., 2009; MARCONI; LAKATOS, 2010; MAROCO, 2010).

Muitas vezes, a amostra é representada por meio de múltiplas variáveis e, para extrair informações relevantes, é necessário recorrer à análise simultânea dessas variáveis para cada indivíduo ou objeto em análise (HAIR et al., 2009). A essa análise denomina-se análise multivariada de dados. Neste estudo, as funções da estatística descritiva como o estudo da frequência, da média, da mediana, do desvio padrão, da variância e da amplitude dos dados foram

utilizadas. Porém, o uso da análise multivariada, por meio de algumas de suas técnicas exploratórias e confirmatórias, como ferramental, para obtenção dos resultados do estudo é o foco desta seção.

4.3 Análise multivariada

A multivariada é uma das áreas da estatística de maior importância, pois busca soluções para problemas relevantes para a sociedade, na investigação científica dos fenômenos estudados e analisados (FERREIRA, 2011). Dessa forma, a partir das informações coletadas e analisadas, é possível que os profissionais de diversas áreas da ciência tenham suporte teórico e quantitativo na tomada de decisões em seus negócios. Há razões que pesquisadores utilizam para justificar a opção por análises multivariadas (HAIR et al., 2009):

a) A utilização de testes estatísticos separados para cada variável podeprovocar erros;
b) As análises univariadas podem ignorar informações importantes contidas nos dados, que muitas vezes são provenientes das correlações entre as variáveis em análise;
c) Algumas diferenças entre as variáveis, quando analisadas separadamente, podem não ser significativas e, quando analisadas em conjunto, podem se revelar significativas.

A análise multivariada é um tipo de análise estatística frequentemente utilizado para extrair conhecimento de grandes conjuntos de dados. Na estatística, o objetivo desse tipo de análise é medir, explicar e prever o grau de relação entre variáveis (HAIR et al., 2009). Uma variável estatística é uma combinação linear de variáveis com pesos determinados, de acordo com a

técnica a ser utilizada. Pode-se definir uma variável estatística de n variáveis ponderadas (x_1, ..., x_n) da seguinte forma (HAIR et al., 2009):

$$\text{Valor da Variável Estatística} = w_1x_1 + w_2x_2 + w_3x_3 + ... + w_nx_n$$

sendo x_i uma variável e w_i o peso dessa variável. Existem várias técnicas de análises multivariadas. A escolha de qual técnica utilizar pode ser feita com base na classificação das variáveis dependentes e independentes, na quantidade de variáveis disponíveis em um modelo estatístico e em como essas variáveis serão medidas.

Em relação à dependência das variáveis, pode ser utilizada uma **técnica de dependência** ou **técnica de interdependência**. As técnicas de dependência são utilizadas quando uma variável é dependente, sendo prevista ou explicada por outras variáveis (variáveis independentes). As técnicas de interdependência são utilizadas quando não se pode classificar as variáveis como dependentes ou independentes, sendo analisadas simultaneamente com o objetivo de encontrar uma estrutura subjacente a elas. Na Figura 4, são mostradas as diferenças entre as técnicas de interdependência (lado esquerdo) e dependência (lado direito). As técnicas de interdependência analisam variáveis V_n a fim de explicá-las, considerando as demais, em uma estrutura F_n. As técnicas de dependência analisam variáveis independentes I_n para descrever o comportamento das variáveis dependentes D_n.

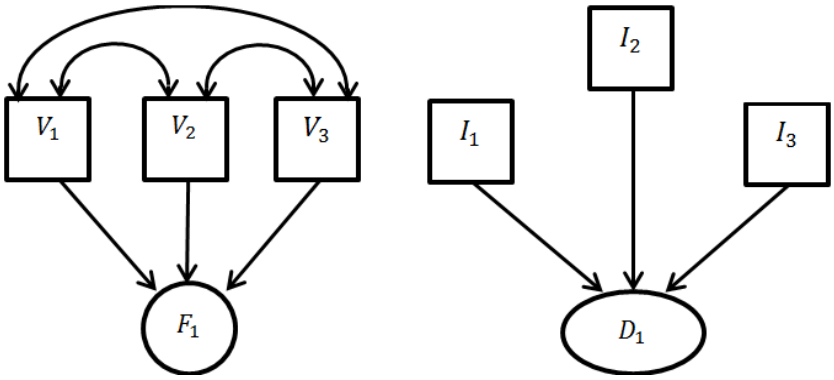


Figura 4 Diferenças entre os objetivos de técnicas de interdependência e dependência

Fonte: Adaptado de Corrêa et al. (2010).

4.4 Tipos de análises multivariadas

Existem várias técnicas de análise multivariada com finalidade bem diversas entre si (HAIR et al., 2009). Além disso, essas técnicas podem ter objetivos **exploratórios** ou **confirmatórios**. Na análise exploratória, o objetivo é definir possíveis relações e utilizar técnicas multivariadas desse caráter para estimá-las (HAIR et al., 2009; HAIR et al., 2014). Na análise confirmatória, o objetivo é utilizar uma técnica multivariada para testar (confirmar) uma relação pré-especificada (HAIR et al., 2009; HAIR et al., 2014).

Segundo informações em pesquisas recentes, a técnica exploratória mais comum para estimar relações entre variáveis é a Análise Fatorial Exploratória (AFE) ou, simplesmente, a Análise Fatorial (AF) (MUHAMMAD; MAQBOOL; ABBASI, 2012; TREIBLMAIER; FILZMOSER, 2010). Para testes confirmatórios, a técnica PLS-SEM tem sido amplamente utilizada pela academia para explicar a variabilidade das variáveis dependentes em um modelo de equações estruturais (CHIN; DIBBERN, 2009; LEE; XIA, 2010; LOWRY; GASKIN, 2014; URBACH; AHLEMANN, 2010). Portanto, para a etapa

exploratória do estudo, foi utilizada a técnica AF. Para a etapa confirmatória do estudo, foi utilizado o PLS-SEM. O referencial teórico, as etapas e os objetivos das técnicas são apresentados nas próximas seções.

4.4.1 Análise fatorial

A Análise Fatorial (AF) é uma técnica de interdependência, cujo propósito principal é definir a estrutura inerente entre as variáveis em análise. Consiste na redução de dados ou de uma estrutura de simplificação, descrevendo, se possível, a razão entre a covariância entre as muitas variáveis em quantidades aleatórias e não observadas chamadas fatores (HAIR et al., 2009; JOHNSON; WICHERN, 2007). Nessa técnica, determina-se qual o conjunto de variáveis observadas compartilha características da variância e da covariância que definem a construção dos fatores (variáveis latentes). Na prática, são coletados dados de variáveis observadas e são utilizadas técnicas analíticas fatoriais para confirmar quais variáveis definem esses construtos ou fatores ou explora-se quais variáveis estão relacionadas aos fatores (RAYKOV; MARCOULIDES, 2012).

No modelo de AF, cada uma das n variáveis representa uma combinação linear de m fatores comuns e de um fator específico. Para o i-ésimo indicador tem-se (JOHNSON; WICHERN, 2007):

$$x_{ij} = \alpha_{i1}f_{1j} + \alpha_{i2}f_{2j} + K + \alpha_{im}f_{mj} + u_i y_{ij} \quad (1)$$

ou,

$$x_{ij} = \sum_{p=1}^{m} \alpha_{ip} f_{pj} + u_i y_{ij} \quad (2)$$

sendo f_{pj} o valor do p-ésimo fator comum para a j-ésima observação, a_{ip} (com p = 1, ... , m) o coeficiente dos fatores comuns, u_i o coeficiente dos fatores específicos e y_{jj} o j-ésimo valor do i-ésimo fator específico, ou seja, é o valor único que representa a parte não explicada pelos fatores comuns.

Em termos lineares, como função dos m fatores comuns f_j, com as quais se relaciona com cargas fatoriais, ou coeficientes de conexão $\alpha_{ij,}$ que indicam em que medida e direção as variáveis em x_i estão relacionadas ao fator f_j, e com um fator único y_i, que responde pela variância remanescente (JOHNSON; WICHERN, 2007). Essas intercorrelações observadas entre as p variáveis podem ser explicadas por um conjunto reduzido de fatores comuns, e por um conjunto de m fatores específicos que explicam a variância total do modelo (MAROCO, 2010). A variância de x_i pode ser dividida em dois componentes (JOHNSON; WICHERN, 2007; MAROCO, 2010):

$$V(x_i) = h_i^2 + \omega_i \quad (3)$$

sendo h_i^2 a comunalidade de x_i, ou seja, a estimativa da variância explicada pelos fatores comuns, e ω_i a porção da variância única de x_i, que se divide em dois componentes: i) a variância específica, que caracteriza a variável; e ii) o erro, que corresponde a porção da variância associada a erros de medição.

Portanto, o modelo fatorial pode ser definido pela equação 4 (JOHNSON; WICHERN, 2007; MAROCO, 2010):

$$x = \alpha f + \beta \quad (4)$$

sendo x o vetor de variáveis padronizadas, f o vetor dos fatores comuns, β o vetor de fatores específicos e α uma matriz de pesos fatoriais.

Após o modelo fatorial definido, é necessário testar a adequação desse modelo. O método de utilização mais geral é o *Kaiser-Meyer-Olkin Measure of*

Sampling Adequacy - Medida de Adequação de Amostragem *Kaiser-Meyer-Olkin* (KMO) (MAROCO, 2010). O KMO é a razão da soma dos quadrados das correlações das variáveis, dividida por essa mesma soma acrescentada da soma dos quadrados das correlações parciais das variáveis (MAROCO, 2010). O valor do KMO varia [0,1], sendo que um valor próximo de 1 indica perfeita adequação dos dados para a AF. Na Tabela 1, são mostrados os valores críticos do KMO.

Tabela 1 Valores críticos do Teste KMO para adequação da AFE

Valor do KMO	**Recomendação da AFE**
]0.9 - 1.0]	Excelente
]0.8 - 0.9]	Boa
]0.7 - 0.8]	Média
]0.6 - 0.7]	Medíocre
]0.5 - 0.6]	Mau, mas ainda aceitável
≤ 0.5	Inaceitável

Fonte: Maroco (2010).

Outro teste de adequação que precede a AF é o *Bartlett Test of Sphericity* (BTS) - Teste de Esfericidade de *Bartlett*, que testa a hipótese da matriz de correlação ser uma matriz identidade (diagonal igual a 1 e as outras medidas iguais a zero). Se a matriz de correlação é uma matriz identidade, significa que não há correlação entre as variáveis (HAIR et al., 2009; MAROCO, 2010). Além disso, nem todos os fatores são aproveitáveis em uma AF. A determinação da quantidade de fatores pode ser facilitada com a utilização de critérios preestabelecidos. A técnica mais comumente utilizada é o critério da raiz latente ou do autovalor (HAIR et al., 2009). Na Figura 5, observa-se o gráfico do autovalor em relação à quantidade de fatores.

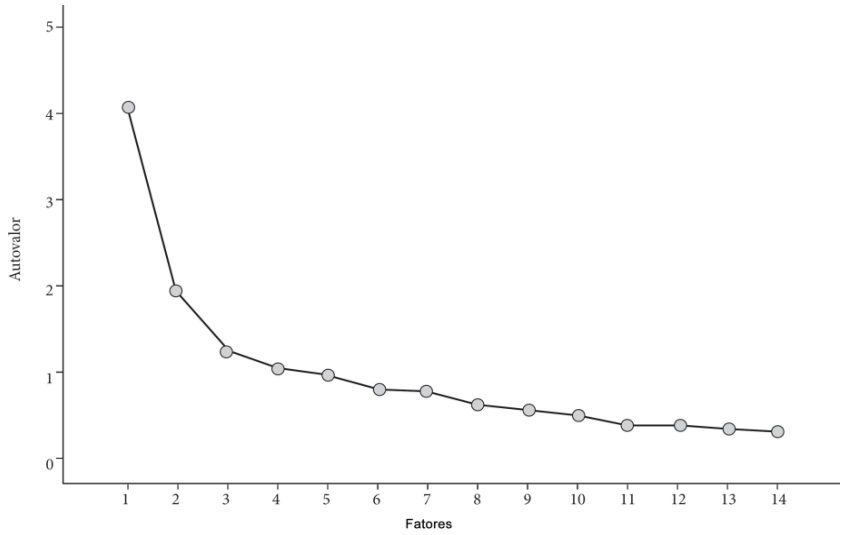


Figura 5 Gráfico de determinação de fatores com o critério do autovalor
Fonte: (MAROCO, 2010).

O raciocínio para o critério da raiz latente é qualquer fator individual explicar a variância de pelo menos uma variável para ser mantido para interpretação no modelo fatorial (HAIR et al., 2009). Logo, apenas os fatores que têm raízes latentes ou autovalores maiores que 1 são considerados significantes. O autovalor é calculado pela soma dos quadrados dos pesos de cada variável, representada pelo fator obtido (HAIR et al., 2009). Para identificar a quantidade ótima de fatores a serem extraídos, é necessário avaliar o ponto de corte do gráfico. O ponto de corte é avaliado conforme o ângulo de inclinação da curva resultante. Quando a curva tende a ficar horizontal (próxima do valor 1), tem-se o indicativo da quantidade máxima de fatores a serem extraídos (HAIR et al., 2009; MAROCO, 2010).

Embora o modelo fatorial na AF consiga produzir múltiplas matrizes α (matrizes de pesos fatoriais) capazes de gerar uma matriz padronizada x, raramente esse modelo resulta em fatores comuns que podem ser interpretados (HAIR et al., 2009). Se a solução não é interpretável, então a matriz α pode ser

multiplicada por outra matriz qualquer ortogonal (HAIR et al., 2009). Assumindo que T é uma matriz ortogonal, tal que TT' = I, sendo I uma matriz identidade, o modelo fatorial pode ser reescrito como na equação 5 (HAIR et al., 2009; MAROCO, 2010):

$$x = (\alpha T)(T'f) + \beta \quad (5)$$

Sendo x designada por rotação de fatores. A matriz α pode ser transladada até ser alcaçada uma solução interpretável, sem que a orientação dos vetores que representam as variáveis seja alterada.

Há dois tipos de rotação: i) **ortogonal**; e ii) **oblíqua**. Na rotação ortogonal, os eixos são mantidos a 90º. Na rotação oblíqua, os eixos são rotacionados sem manter o eixo de 90º entre os eixos de referência. Na prática, na rotação, o objetivo é simplificar as linhas e as colunas da matriz fatorial para facilitar a interpretação. A simplificação de linhas significa tornar os valores máximos em cada linha próximos de zero (ou seja, maximizar a carga de uma variável em um único fator). A simplificação de colunas significa tornar os valores máximos em cada coluna próximos de zero (ou seja, tornar a quantidade de cargas "elevadas" a menor possível). O critério de rotação mais utilizado na rotação é o *Varimax* (para rotação ortogonal), cujo objetivo é "espalhar" os quadrados das cargas de cada fator o máximo possível. Portanto, espera-se encontrar grupos de coeficientes grandes e coeficientes desprezíveis em qualquer coluna da matriz de cargas rotacionadas (HAIR et al., 2009; MAROCO, 2010).

O último estágio da análise fatorial corresponde à estimação dos valores dos fatores comuns para cada indivíduo como função das variáveis observadas. Na literatura, há oito métodos conhecidos para extrair fatores em uma AF (HAIR et al., 2009):

a) **Análise de Componentes Principais.** É um método no qual os fatores estão baseados na variância total. O primeiro componente tem a maior variância comum, componentes sucessivos possuem mais variâncias específicas e variância do erro. É o método mais adequado quando se quer extrair a maior proporção da variância com o menor número de fatores;

b) **Máxima Verossimilhança.** Esse método produz estimativas dos parâmetros as mais prováveis de ter produzido as correlações observadas. Entretanto, para utilizar esse método, a amostra utilizada deve ter sido originada de uma população com distribuição normal multivariada;

c) **Fatoração pelo Eixo Principal.** É o método que utiliza o quadrado da correlação múltipla como estimativa das comunalidades. As comunalidades são colocadas na diagonal da matriz principal antes da extração dos fatores;

d) **Mínimos Quadrados Não-Ponderados.** Esse método minimiza o quadrado da diferença entre as matrizes de correlações observadas e reproduzidas;

e) **Mínimos Quadrados Generalizados.** Assim como o método dos Mínimos Quadrados Não-Ponderados, minimiza o quadrado da diferença das matrizes de correlações observadas e reproduzidas. Entretanto, pondera o resultado pelos valores dos itens envolvidos na análise;

f) **Fatoração Alfa.** Nesse método, os itens em análise são tratados como uma amostra da população dos possíveis itens. Os fatores são selecionados com o objetivo de maximizar o coeficiente alfa de confiabilidade;

g) **Fatoração pela imagem.** Esse método é baseado no conceito de uma imagem de um item. Esse item é entendido como uma variável (dependente) de uma regressão múltipla em que os demais itens seriam as variáveis independentes;
h) **Mínimo Residual.** Nesse método, os fatores são extraídos da matriz de correlação, ignorando os elementos da diagonal.

As técnicas analíticas fatoriais podem atingir seus objetivos de uma **perspectiva exploratória** ou de uma **perspectiva confirmatória**. Sob a perspectiva exploratória, considera-se que os dados oferecem e não estabelecem restrições a *priori* sobre a quantidade de componentes a serem extraídos. A utilidade da perspectiva exploratória está em buscar estruturas em um conjunto de variáveis ou uma forma ou método de reduzir dados; é descobrir a natureza da estrutura subjacente entre as variáveis indicadoras (HAIR et al., 2009). No entanto, se há ideias preconcebidas sobre a estrutura dos dados a ser analisada, seja baseada em suporte teórico ou em pesquisas anteriores, a AF deve assumir a perspectiva confirmatória. Na perspectiva confirmatória, o objetivo é avaliar o grau em que os dados satisfazem a estrutura esperada, por meio de teste de hipóteses sobre quais variáveis deveriam ser agrupadas em um fator ou da quantidade exata de fatores (HAIR et al., 2009). Nessa perspectiva, a teoria vem em primeiro lugar, o modelo é derivado da teoria e, depois, o modelo é testado para obter a consistência com os dados observados, usando uma abordagem de Modelagem de Equações Estruturais (*Structural Equation Modeling* - SEM).

Assim, a questão é se o modelo produz uma matriz de covariância populacional[5] consistente com a matriz de covariância amostral[6] (observada). Se

[5] É uma matriz que representa caminhos estatisticamente significantes estimados no modelo estrutural (HAIR et al., 2009).

o modelo for bom, os parâmetros estimados produzirão uma matriz de covariância populacional estimada próxima à matriz de covariância amostral. Essa "proximidade" é avaliada primeiramente pelo teste qui-quadrado e, posteriormente, pelos índices de ajuste (RAYKOV; MARCOULIDES, 2012). A aplicação da AF segue a seguinte estrutura (FIELD, 2013; HAIR et al., 2009; JOHNSON; WICHERN, 2007; TREIBLMAIER; FILZMOSER, 2010):

a) **Entrada de Dados**: um conjunto de valores de variáveis para cada objeto ou indivíduo na amostra;
b) **Cálculo das Correlações entre as Variáveis**: a partir da entrada dos dados, obtém-se a matriz de correlações. Para o cálculo dessa matriz, podem ser utilizadas as abordagens **Análise Fatorial R**, cujo objetivo é agrupar as diferentes variáveis em alguns fatores específicos e **Análise Fatorial Q**, cujo objetivo é formar grupos de casos com base em sua similaridade em um conjunto de características;
c) **Extração Inicial dos Fatores**: há diferentes métodos de extração de fatores da matriz de correlações, cujo objetivo é encontrar um conjunto de fatores que formem uma combinação linear das variáveis originais ou da matriz de correlações. Dessa forma, se as variáveis x_1, x_2, ..., x_n são altamente correlacionadas entre si, elas são combinadas para formar um fator; de forma análoga, pode-se fazer com as demais variáveis da matriz de correlação. A primeira descoberta é chamada de primeiro fator principal. Em seguida, a variância explicada pelo primeiro fator é subtraída da matriz de

[6] É uma matriz típica de entrada para a estimação do modelo composta das variâncias e covariâncias observadas para cada variável medida (HAIR et al., 2009).

correlações original, resultando em matrizes residuais, que formam os fatores posteriores;

d) **Rotação da Matriz**: a matriz de fatores iniciais, que indica a relação entre as variáveis estudadas, raramente resulta em fatores que podem ser interpretados. No entanto, uma análise torna-se possível e útil por causa da sua capacidade para produzir fatores interpretáveis, por meio de métodos de rotação da matriz. Essa rotação transforma a matriz de fatores em uma matriz rotacionada, maximizada, significativa, mais simples e mais fácil de interpretar. A ideia básica da rotação é identificar fatores que possuam variáveis com alta correlação e outros com variáveis que possuam baixa correlação;

e) **Interpretação dos Fatores:** Como resultado da etapa **Rotação da Matriz**, têm-se a quantidade de fatores extraídos, que variáveis originais fazem parte de cada fator encontrado, quais seriam os fatores gerados pela análise e qual o seu grau de importância para a construção de um modelo fatorial.

4.4.2 Modelagem de Equações Estruturais (*Structural Equation Modeling* - SEM) e *Partial Least Squares* (PLS)

Após a execução AF, é obtido como resultado um conjunto de fatores que explicam a correlação entre variáveis. Porém, não se sabe se os fatores são correlacionados entre si. Para tanto, é necessário modelar estatisticamente essas hipóteses por meio de uma técnica chamada **Modelagem de Equações Estruturais** (*Structural Equation Modeling* - SEM). Essa técnica é uma família de modelos estatísticos que buscam explicar as relações entre múltiplas variáveis ou construtos, expressas em uma série de equações, descrevendo-as por meio da análise dos indicadores a serem medidos (HAIR et al., 2009; JOHNSON;

WICHERN, 2007). Embora essas relações entre variáveis fossem exploradas, a maneira em que os fatores se relacionam precisa ser testada e validada estatisticamente. As técnicas mais comuns para estimação e teste do SEM são a **Análise Fatorial Confirmatória** (AFC) e o ***Partial Least-Square*** (PLS) (BROWN, 2006).

A AFC é utilizada para fornecer um teste confirmatório do modelo de equações estruturais, derivado dos construtos ou fatores de uma AF. Entretanto, outras técnicas, denominadas técnicas da segunda geração das análises multivariadas, vêm sendo utilizadas para testar e confirmar modelos estatísticos, como o PLS (BROWN, 2006; HAIR et al., 2014). No PLS, são fornecidas estimativas mais confiáveis e precisas das relações, mesmo quando existem problemas que podem impedir uma solução de ser validada em SEM (HAIR et al., 2009; HAIR et al., 2014). Essa técnica possui uma estrutura semelhante ao SEM, com estimativas paramétricas de variáveis, mas difere em alguns aspectos, por exemplo, (i) trata os fatores como índices compostos individuais, (ii) encontra soluções baseadas na variância mínima entre os construtos e as variáveis independentes, (iii) não exige boas características de medição para obter resultados e (iv) é menos sensível ao tamanho amostral (HAIR et al., 2009; HAIR et al., 2014; JOHNSON; WICHERN, 2007).

O PLS foi desenvolvido na década de 60, por Herman Wold, como uma técnica econométrica, que busca entender a relação entre variáveis econômicas aplicando um modelo matemático (TOBIAS, 1995). É uma técnica utilizada para estimar de forma sensível os coeficientes de um sistema de equações estruturais, com o método dos mínimos quadrados (HAIR et al., 2009; JOHNSON; WICHERN, 2007; TOBIAS, 1995). A técnica PLS vem ganhando interesse e uso entre pesquisadores de software nos últimos anos (CHIN; DIBBERN, 2009; LEE; XIA, 2010; LOWRY; GASKIN, 2014; URBACH; AHLEMANN, 2010),

por causa de sua capacidade de modelar os construtos latentes em condições de não normalidade e de amostras de pequeno ou médio tamanho.

Sendo uma técnica de Modelagem de Equações Estruturais (*Structural Equation Modeling* - SEM) baseada em componentes, o PLS é semelhante à regressão, mas modela simultaneamente os caminhos estruturais (as relações teóricas entre as variáveis latentes) e os caminhos de medição (as relações entre a variável latente e seus indicadores) (JOHNSON; WICHERN, 2007; SEBER, 2004). Ao invés de assumir pesos iguais para os indicadores de escala, o PLS permite variação em cada indicador, contribuindo para a formação do valor final, composto da variável latente (JOHNSON; WICHERN, 2007; SEBER, 2004). Assim, os pesos inferiores são dados para indicadores com relacionamentos mais fracos e os pesos superiores são dados para os indicadores relacionados e da construção latente (JOHNSON; WICHERN, 2007; SEBER, 2004).

A Modelagem de Equações Estruturais com a técnica PLS (PLS-SEM - *Partial Least Squares Structural Equation Modeling*) consiste em dois elementos (HAIR et al., 2014):

a) **Modelo estrutural**: também denominado de modelo interno no contexto do PLS-SEM, representa os construtos. No modelo estrutural, são mostrados os relacionamentos entre os construtos;
b) **Modelo de medição**: também referido como modelo externo no contexto do PLS-SEM, representa as relações entre os construtos e as variáveis indicadoras. Há dois tipos de modelos de medição, um para variáveis latentes endógenas (construtos explicados no modelo) e outro para as variáveis latentes exógenas (construtos que explicam outros construtos no modelo). Esse modelo pode ser representado pela equação 6:

$$x_i = x_j + \varepsilon \quad (6)$$

Sendo x_i o valor medido em uma escala contínua para uma simples medida ou observação, x_j o valor verdadeiro da variável (variável latente) para a observação e ε o erro da medida, também observado.

Dependendo da forma em que o SEM é desenhado, uma variável pode tecnicamente ser uma variável independente (que representa uma causa), ou uma variável dependente (que representa um resultado ou efeito) nas diferentes partes do modelo; se uma variável tem caminho que conduz a outra variável, ela é classificada como variável latente endógena. Na Figura 6, é ilustrado um exemplo de um SEM simples. Nesse modelo estrutural, cada construto do **Modelo Externo** possui três indicadores (variáveis latentes exógenas que explicam o construto). Por exemplo, o construto **Variável Independente "A"** pode ser explicado pela correlação entre as variáveis **Indicador 1**, **Indicador 2** e **Indicador 3**. A linha retilínea entre os construtos **Variável Independente "A"** e **Variável Dependente** mostra uma relação de dependência entre os construtos. Além disso, as variáveis **Indicador 7**, **Indicador 8** e **Indicador 9** possuem relação direta com o construto **Variável Dependente**.

O primeiro passo de pré-execução do teste do modelo é a execução de um procedimento chamado *bootstraping*. O *bootstraping* é aplicado para testar se os coeficientes dos pesos das variáveis e os coeficientes de correlação entre os construtos do modelo proposto são significativos (HAIR et al., 2014). O PLS-SEM não assume que os dados são normalmente distribuídos, o que implica que os testes paramétricos de significância. Em vez disso, o PLS-SEM depende de um procedimento de *bootstraping* não paramétrico (DAVISON; HINKLEY, 1997; EFRON; TIBSHIRANI, 1993) para testar a significância dos coeficientes do modelo.

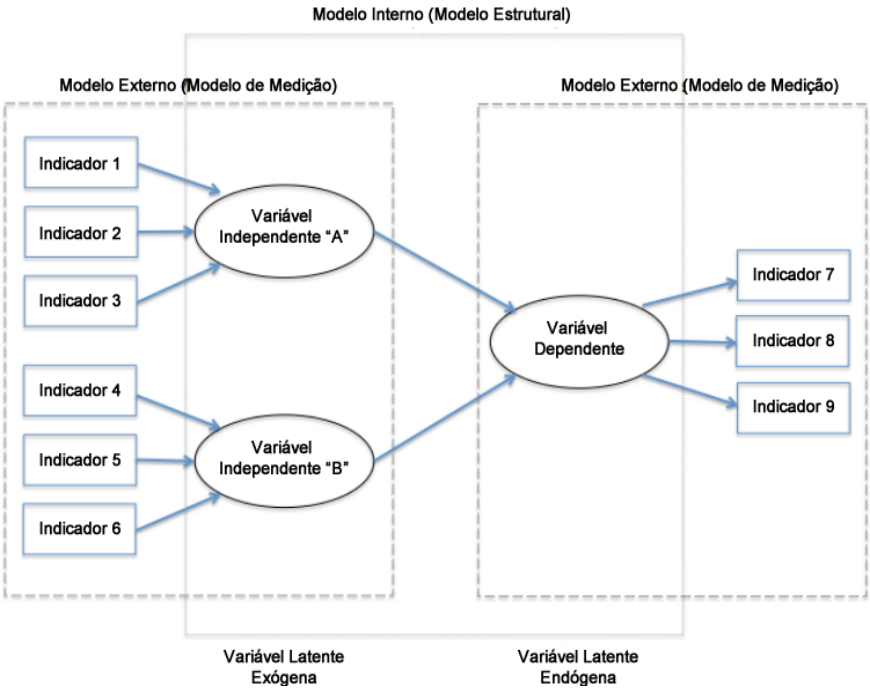


Figura 6 Modelo interno x modelo externo em um diagrama PLS-SEM
Fonte: Adaptado de Kwong e Wong (2013).

Conforme explicado na Figura 7, as subamostras são criadas com observações aleatórias, a partir do conjunto original de dados. A subamostra é utilizada para estimar as correlações do modelo. Recomenda-se como parâmetro 5.000 observações (HAIR et al., 2014). Este processo é repetido até criar uma grande quantidade de subamostras aleatórias, correspondente ao tamanho real da amostra. As estimativas dos parâmetros (por exemplo, os pesos das variáveis e os coeficientes das correlações entre os construtos) calculados a partir das subamostras são utilizadas para derivar os erros padrão para as estimativas. Com essa informação, os valores p são calculados para avaliar a significância estimada (HAIR et al., 2014).

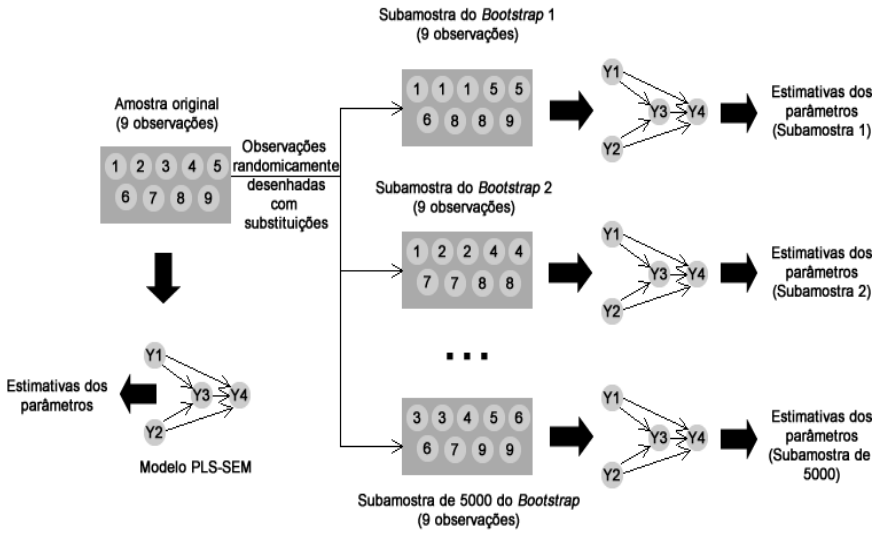


Figura 7 Método de *bootstraping* no PLS-SEM
Fonte: Adaptado de Hair et al. (2014).

Em seguida, a estimação no PLS é realizada iterando sobre uma sequência de partes do modelo estrutural com o objetivo de verificar a capacidade de previsão do modelo e as relações entre os construtos. Essa avaliação determina o quão os dados empíricos coletados se ajustam à teoria. As partes do modelo constituem-se no modelo de medição para as variáveis latentes, chamadas de indicadores, e no conjunto de relacionamentos que conectam essas variáveis latentes (construtos). Essas partes são chamadas respectivamente de "modelos externos" (*outer model*) e de "modelos internos" (*inner model*) (HAIR et al., 2014; JOHNSON; WICHERN, 2007). A etapa inicial é avaliar os níveis significativos de colinearidade entre os construtos do modelo estrutural, pois os coeficientes de caminho são baseados em regressões PLS (*Partial Least Squares* - Mínimos Quadrados Parciais) de cada variável latente endógena nos construtos predecessores correspondentes. Na modelagem por mínimos quadrados parciais, as matrizes X e Y são decompostas simultaneamente em uma soma de A variáveis latentes. A matriz das variáveis

independentes X e a matriz das variáveis dependentes Y são representadas pelos *scores* e pelos pesos, conforme as equações 7 e 8 (BEEBE; KOWALSKI, 1987; GELADI; JOHNSON; WICHERN, 2007; KOWALSKI, 1986; SEBER, 2004):

$$X = TP^t + \varepsilon_x = \sum t_A p_A^T + \varepsilon_x \quad (7)$$

$$Y = UQ^t + \varepsilon_y = \sum t_A p_A^T + \varepsilon_y \quad (8)$$

Sendo X a matriz de dados (medida instrumental), Y a matriz de respostas da propriedade de interesse, T e U os *scores* para as duas matrizes de dados, P e Q os respectivos *loadings* das matrizes X e Y, A a quantidade de variáveis latentes e ε_x e ε_y os respectivos resíduos compostos pelas variáveis latentes descartadas, ou seja, as matrizes que contêm a parte não modelada. É necessário achar a melhor quantidade de variáveis latentes, o que normalmente é feito usando um procedimento chamado validação cruzada ("*cross validation*"), no qual o erro mínimo de previsão é determinado (BEEBE; KOWALSKI, 1987; GELADI; KOWALSKI, 1986).

O modelo estrutural no PLS-SEM é avaliado com base em critérios heurísticos determinados pela capacidade de previsão do modelo. Os principais critérios para a avaliação do modelo estrutural no PLS-SEM são a significância dos coeficientes de caminhos, os valores do coeficiente de determinação R^2, o efeito f^2, a relevância preditiva (Q^2) e o efeito q^2. De forma específica, a interpretação dos efeitos dessas variáveis sobre as outras é feita examinando os coeficientes ou os pesos para cada variável na relação.

4.5 Considerações finais

Nesta seção, foram discutidas técnicas utilizadas por pesquisadores da área para executar análise quantitativa de dados. O conceito de análise multivariada, que consiste na análise de múltiplas variáveis e as relações entre elas foi apresentado. A multivariada é uma das áreas da estatística de maior importância, pois busca soluções para problemas relevantes para a sociedade na investigação científica dos fenômenos estudados e analisados. A partir das informações coletadas e analisadas pelas técnicas multivariadas existentes, torna-se possível para profissionais de diversas áreas da ciência ter suporte teórico e quantitativo na tomada de decisões em seus negócios.

As técnicas AF e PLS-SEM têm sido largamente difundidas e utilizadas por pesquisadores para a validação de aspectos relacionados ao conjunto de variáveis estudado neste trabalho. Essas técnicas foram descritas nesta seção. Na Seção 5, o estado da arte da qualidade interna de software é investigado. Os resultados da seção 5 formaram a base teórica do estudo, cujos dados coletados formaram a base de dados analisada pela estatística multivariada descrita nesta seção.

5 QUALIDADE INTERNA DE SOFTWARE - ESTADO DA ARTE

5.1 Considerações iniciais

Embora a tecnologia orientação a objetos (OO) esteja consolidada no desenvolvimento de sistemas de software, formas eficientes de medir e de caracterizar a qualidade interna desses sistemas ainda são discutidas na literatura (BAILEY et al., 2007). Estudos da área Engenharia de Software têm focado em entender como medidas podem caracterizar a qualidade desses sistemas (ANDA, 2007).

Dentre os temas abordados, estão fatores de propensão a falhas (AL DALLAL, 2012; BRIAND et al., 2000), produtividade ou esforço em manutenção (CHIDAMBER; DARCY; KEMERER, 1998; FERREIRA et al., 2012) e reúso/refatoração de software (SOUZA; MAIA, 2013). Nesses estudos, diversos modelos para medição da qualidade em sistemas de software OO têm sido propostos (ORENYI; BASRI; JUNG, 2012), cujos resultados são obtidos utilizando técnicas para análise inferencial e análise multivariada de dados. Embora os objetivos desses estudos sejam diversos, pouco se sabe sobre quais medidas são mais relevantes e quais técnicas são mais adequadas para caracterizar a qualidade em determinado aspecto de qualidade, por exemplo, a Manutenibilidade. A identificação desses estudos foi realizada por meio de uma Revisão Sistemática de Literatura (RSL).

Uma RSL é uma técnica que visa identificar e interpretar estudos disponíveis e relevantes para uma questão de pesquisa específica (KITCHENHAM, 2004). Essa técnica é popular e bem estabelecida na área médica, mas tem sido extensamente utilizada na área Engenharia de Software (HALL et al., 2012; KITCHENHAM, 2004). Nesta seção, o objetivo é utilizar a RSL para identificar estudos sobre o estado da arte da avaliação da qualidade

interna de sistemas de software OO, utilizando técnicas/modelos estatísticos. O escopo da RSL seguiu protocolo, cujo resultado foi de artigos científicos de eventos e de conferências entre 2004 e 2014.

Esta seção está organizada da seguinte forma. A técnica RSL é brevemente explicada na subseção 5.2. A metodologia da RSL é apresentada na subseção 5.3. O resultado da revisão é analisado na subseção 5.4. Outros estudos relacionados à proposta, previamente encontrados por meio de uma revisão de literatura, são apresentados na subseção 5.5. Ameaças à validade do estudo são apresentadas na subseção 5.6.

5.2 Revisão sistemática de literatura

A Revisão Sistemática de Literatura (RSL) é uma técnica para identificar, avaliar e interpretar estudos ou pesquisas disponíveis e relevantes para uma questão de pesquisa específica, área temática ou fenômeno de interesse. Os estudos individuais empíricos diretamente relacionados à questão em pesquisa são chamados estudos primários. Estudos secundários visam revisar, investigar e sintetizar as evidências identificadas nos estudos primários (KITCHENHAM, 2004). Dessa forma, RSL pode ser considerada um estudo secundário. A aplicação da RSL segue um roteiro com estratégias e critérios pré-definidos de seleção e de análise de documentos, organizados em três fase (ATALLAH; CASTRO, 1997; HALL et al., 2012; KITCHENHAM, 2004; ZHANG; BABAR, 2011):

a) **Planejamento**: o motivo da realização da RSL é estabelecido. Seus tópicos são: i) **Descrição da Pesquisa:** motivações e objetivos para a pesquisa; ii) **Definição de Questões de Pesquisa**: questões relacionadas às motivações da pesquisa; iii) **Desenvolvimento do**

Protocolo: protocolo a ser aplicado às buscas; e iv) **Avaliação do Protocolo**: avaliação da aplicação da RSL;

b) **Execução**: a investigação em fontes definidas na fase anterior é realizada. O estudo e a classificação dos trabalhos encontrados podem ser feitos, guiados pelos critérios de inclusão e de exclusão. Seus tópicos são: i) **Obtenção das Pesquisas:** busca nas fontes definidas, organizando os resultados por algum critério para facilitar as próximas etapas; ii) **Seleção Primária:** primeira seleção dos resultados obtidos (título, palavras-chave e resumo dos trabalhos são lidos para verificar atendimento dos critérios); iii) **Seleção Secundária**: segunda seleção realizada, cujo objetivo é eliminar resultados não apropriados; os artigos são lidos e verificados quanto ao atendimento dos critérios; e iv) **Organização dos Resultados:** artigos selecionados são organizados para serem analisados;

c) **Análise dos Resultados**: é a coleta e a organização dos dados extraídos dos artigos selecionados. O resultado é analisado de maneira global, gerando melhor planejamento, caso necessário. Seus tópicos são: i) **Coleta e Organização dos Dados**: resultados obtidos na Seleção Secundária são lidos e interpretados; os dados são extraídos e organizados em forma de um relatório, cujo conteúdo responde as questões de pesquisa definidas no protocolo; e ii) **Avaliação dos Resultados**: o relatório é revisado para buscar a publicação dos resultados.

5.3 Metodologia

A metodologia do trabalho segue o protocolo da RSL. Esses artefatos são apresentados nas próximos itens.

5.3.1 Questões de pesquisa

O objetivo é identificar estudos sobre avaliação da qualidade interna de sistemas de software OO, utilizando técnicas/modelos estatísticos. As seguintes questões de pesquisa foram formuladas:

Q1: Quais são as principais medidas investigadas, na última década, para avaliar a qualidade interna de sistemas de software orientados a objeto?

Q2: Quais são os principais técnicas/modelos estatísticos utilizados, na última década, por pesquisadores da área Engenharia de Software, para avaliar a qualidade interna de sistemas de software orientados a objeto?

5.3.2 Seleção de fontes e *string* de busca

Para execução da RSL, foram selecionadas como fontes de busca, ferramentas de busca na *web*, destinadas à busca extensiva de textos científicos completos e metadados. A seleção das fontes foi conduzida seguindo os critérios:

a) Deveria conter a opção de pesquisa avançada com utilização de palavras-chave;
b) Filtragem dos resultados por ano de publicação e área e/ou tipo de publicação;
c) Exportação do resultado da consulta em formato *BibTex*;
d) Invariabilidade no resultado da busca quando utilizado o mesmo conjunto de palavras-chave.

A partir destas condições, foram escolhidas as seguintes fontes de pesquisa:

a) IEEE (http://ieeexplore.ieee.org/);
b) Scopus (http://www.scopus.com);
c) ScienceDirect (http://www.sciencedirect.com);
d) Springer (http://link.springer.com/advanced-search);
e) Ei Compendex (http://www.engineeringvillage.com/).

Dentre as fontes de busca pré-selecionadas, inicialmente, a ACM Digital Library foi considerada como repositório a ser utilizado. Porém, apresentou navegabilidade insatisfatória e não disponibiliza opção de exportação do resultado da consulta. Assim como a Springer, sua máquina de busca disponibiliza opção de exportação por resultado, mas a exportação de alguns documentos possui falhas, por exemplo, falta do resumo (*abstract*) e arquivos vazios. Em um estudo realizado sobre as máquinas de busca (BAILEY et al., 2007), constatou-se que resultados fornecidos pela máquina de busca da ACM são inconsistentes em algumas situações.

Para realizar as buscas, foi utilizada a *string* de busca, construída com base nas palavras-chave e nos sinônimos definidos no protocolo da pesquisa:

(software **OR** application **OR** applications **OR** system **OR** systems **OR** program **OR** programs **OR** “software system” **OR** “software systems”)

AND

(metric **OR** metrics **OR** “software metrics” **OR** “code metrics” **OR** measure **OR** measures **OR** measurement **OR** measuring)

AND

(quality **OR** “internal quality” **OR** “code quality” **OR** “software quality”)

AND

(“object oriented” **OR** “object-oriented” **OR** “oo”)

AND

(“statistics” **OR** “statistical analysis” **OR** “statistical analyses” **OR** “statistical technique” **OR** “statistical techniques” **OR** “statistical approach” **OR** “statistical approaches” **OR** “statistical tools” **O**R “statistical method” **OR** “statistical methods” **OR** “statistical model” **OR** “statistical models” **OR** “quantitative analysis”)

5.3.3 Critérios de inclusão e de exclusão

Critérios para a inclusão ou exclusão de trabalhos foram definidos. Nos critérios de inclusão, foram considerados:

a) Artigos completos publicados;
b) Ser da área de ciência da computação;
c) Ter sido publicado entre os anos 2004 e 2014, inclusive;
d) Ser artigo de *Journal* ou *Conference Paper*;
e) Ter sido publicado no idioma Inglês;

f) Apresentar estudo sobre a avaliação da qualidade interna de sistemas de software OO, utilizando medidas para quantificar essa qualidade, e técnicas/modelos estatísticos para a avaliação dos resultados.

Nos critérios de exclusão, foram desconsiderados trabalhos:

a) Texto de acesso restrito;
b) Textos incompletos;
c) Artigos *In press* (ainda não publicados efetivamente);
d) Não artigos (por exemplo, *Table of contents*, *Index* e *Standards*);
e) Textos duplicados;
f) Não atendem o critério de inclusão.

Três pesquisadores (PA, PB e PC) foram envolvidos na seleção e realizaram o seguinte procedimento:

a) PA executou a *string* de busca nas fontes selecionadas e documentou os resultados no JabRef[7];
b) PA verificou e excluiu os não artigos e artigos repetidos. Na identificação de artigos repetidos, foram mantidos aqueles com palavras-chave que melhor o descreviam;
c) PA e PB avaliaram os artigos encontrados quanto ao atendimento dos critérios de inclusão e de exclusão e documentaram os resultados. Essa avaliação foi realizada por meio da leitura do título, do resumo e das palavras-chave. Os artigos, cujas avaliações causaram dúvidas, foram incluídos;

[7] http://jabref.sourceforge.net/

d) Realizou-se a interseção entre os artigos selecionados por PA e PB, sendo documentados. Desacordos foram resolvidos entre PA e PB. Em casos que não houve consenso, o artigo foi incluído. Os artigos excluídos foram documentados em uma lista com justificativa;
e) PC avaliou os artigos resultantes da *string* de busca considerando título, resumo e palavras-chave e realizou a interseção entre os artigos selecionados por ele e os selecionados por PA e PB. PA, PB e PC resolveram discrepâncias. O resultado foi o conjunto de artigos resultantes do mapeamento;
f) O texto completo dos artigos resultantes foi lido na íntegra e os dados foram extraídos. Artigos que não atendiam aos critérios de inclusão foram descartados.

5.4 Resultados da revisão

Nesta subseção, os resultados da RSL são apresentados (Tabela 2) e discutidos. Para as fontes de pesquisa, a mesma *string* de busca construída foi utilizada, adequando-a às restrições da máquina de busca das fontes. Foram encontrados 8.231 trabalhos distribuídos em 6.631 trabalhos no IEEE (80,6%), 33 trabalhos no Compendex (0,4%), 292 trabalhos no Springer (3,5%), 293 trabalhos no ScienceDirect (3,6%) e 982 trabalhos no Scopus (11,9%).

Tabela 2 Seleção Primária

Fontes	**Total**	**Não Artigos**	**Duplicados**	**Excluídos**	**Incluídos**
IEEE	6.631	501	18	6.082	30
Compendex	33	1	1	25	6
Springer	292	4	0	282	3
ScienceDirect	293	6	0	285	3
Scopus	982	6	0	914	37
Total	8.231	518	19	7.633	79

A maior quantidade de trabalhos foi obtida no IEEE, sendo 501 trabalhos do tipo *Table of Contents*, *Index* e *Standards* (7,5%), 18 trabalhos duplicados (0,2%) e 6.082 trabalhos irrelevantes (91,3%). Ao final, o resultado foi de 30 trabalhos relevantes (0,9%). No repositório da IEEE, a busca foi filtrada pelo tipo de publicação (*Conference Publications*, *Journals & Magazines* e *Standards*), ano de publicação e tópicos, mas não possibilita a seleção das opções simultaneamente. A seleção das opções, segundo os critérios de inclusão, foi feita manualmente e os resultados foram atualizados conforme a aplicação de cada filtro estabelecido.

No Compendex, a busca foi realizada na opção *Expert Search*. Foram utilizados os filtros *Limit To* (com período de "2004 a 2014", inclusive), *Document Type* (com valores *Conference Article* e *Journal Article*) e *Language* (com o valor *English*). Os resultados foram um trabalho do tipo *Table of Contents* (3%), um trabalho repetido (3%), 25 trabalhos considerados irrelevantes (63,7%) e seis trabalhos relevantes (30,3%).

Na Springer, a máquina de busca retornou quatro trabalhos repetidos (0,6%), 282 trabalhos irrelevantes (93,1%) e seis trabalhos relevantes (6,3%). A busca foi realizada com a opção *Advanced Search*, inserindo a *string* de busca nos campos "*with at least one of the words*" e "*where the title contains*". Além disso, foi desmarcado o item *Include Preview-Only content* para remover artigos *In-Press*. Os filtros utilizados foram *Content Type* (com valor *Article*), *Discipline* (com valor *Computer Science*) e *Language* (com o valor *English*).

A busca no ScienceDirect foi realizada na opção *Expert Search*, selecionando a opção *Journals*. O campo *Articles in press* foi desmarcado. Foram utilizados os filtros *Sources* (com valor *All Journals*), *Subject* (com valor *Computer Science*), *Limit by document type* (com valores *Articles*, *Review Article* e *Short Survey*), *Date Range* (com período de "2004 a 2014", inclusive) e *Topic* (com os valores *software*, *computer science*e *information system*). A

máquina de busca retornou seis trabalhos do tipo *Table of Contents* e *Index* (2%), 285 trabalhos irrelevantes (96%) e dois trabalhos relevantes (2%).

No Scopus, a máquina de busca retornou seis trabalhos repetidos (0,6%), 914 trabalhos irrelevantes (93,1%) e 37 trabalhos relevantes (6,3%). Os filtros utilizados foram *Year* (com publicação entre os anos 2004 e 2014, inclusive); *Subject Area* (com valor *Computer Science*), *Document Type* (com valores *Conference Paper* e *Articles*), *Source Type* (com valores *Conference Proceedings* e *Journals*) e *Language* (com o valor *English*).

Os 79 trabalhos (artigos) selecionados (1,6%) como estudos primários são apresentados na Tabela 3. Além das informações ano, título e fonte de obtenção do artigo, há uma coluna (Ref.) que corresponde à referência do artigo.

Tabela 3 Artigos da seleção primária

#	Ano	Título	Fonte	Ref.
1	2004	A Comparison of Cohesion Metrics for Object-Oriented Systems	Scopus	(Etzkorn et al., 2004)
2	2006	A Complexity Metrics Set for Large-Scale Object-Oriented Software Systems	IEEE	(Ma et al., 2006)
3	2013	A Fuzzy Classifier Approach to Estimating Software Quality	Scopus	(Pizzi, 2013)
4	2010	A Hybrid Set of Complexity Metrics for Large-Scale Object-Oriented Software Systems	Springer	(Ma et al., 2010)
5	2007	A Large-Scale Empirical Comparison of Object-Oriented Cohesion Metrics	IEEE	(Barker; Tempero, 2007)
6	2004	A Methodology for Constructing Maintainability Model of Object-Oriented Design	IEEE	(Kiewkanyaet al., 2004)
7	2011	Analysis of Quality of Object Oriented Systems Using Object Oriented Metrics	IEEE	(Kayarvizhy; Kanmani, 2011)
8	2006	Analyzing the Software Quality Metrics for Object Oriented Technology	Scopus	(Parthasarathy; Anbazhagan, 2006)
9	2011	An Analysis of SNA Metrics on the Java Qualitas Corpus	Scopus	(Tonelli et al., 2011)
10	2011	An Empirical Analysis of Lack of Cohesion Metrics for Predicting Testability of Classes	Scopus	(Badri et al., 2011)
11	2005	An Empirical Analysis of Object-Oriented Metrics for Java Technologies	IEEE	(Farooq et al.,2005)

"Tabela 3, continuação"

#	Ano	Título	Fonte	Ref.
12	2005	An Empirical Exploration of the Distributions of the Chidamber and Kemerer Object-Oriented Metrics Suite	Scopus	(Succi et al., 2005)
13	2007	An Empirical Study of Class Sizes for Large Java Systems	IEEE	(Zhang et al., 2007)
14	2007	An Empirical Study of Slice-Based Cohesion and Coupling Metrics	Scopus	(Meyers; Binkley, 2007)
15	2013	An Empirical Study of Source Level Complexity	IEEE	(Xiao, 2013)
16	2011	An Empirical Study on Object-Oriented Metrics and Software Evolution in Order to Reduce Testing Costs by Predicting Change-Prone Classes	IEEE	(Eski; Buzluca, 2011)
17	2008	An Empirical Validation of Object-Oriented Class Complexity Metrics and Their Ability to Predict Error-Prone Classes in Highly Iterative, or Agile, Software: A Case Study	Compendex	(Olague et al., 2008)
18	2008	An Empirical Validation of Object-Oriented Design Metrics for Fault Prediction	Scopus	(Xuet al., 2008)
19	2011	An Empirical Verification of Software Artifacts Using Software Metrics	Scopus	(Shatnawi; Alzu'bi, 2011)
20	2004	An Exploratory Study of Object-Oriented Software Component Size Determinants and the Application of Regression Tree Forecasting Models	Science Direct	(Pendharkar, 2004)
21	2012	An Investigation of Design Level Class Cohesion Metrics	Scopus	(Kaur; Singh, 2012)
22	2010	An Object-Oriented High-Level Design-Based Class Cohesion Metric	Scopus	(Al Dallal; Briand, 2010)
23	2012	A Precise Method-Method Interaction-Based Cohesion Metric for Object-Oriented Classes	Compendex	(Al Dallal; Briand, 2012)
24	2007	A Probabilistic Approach to Predict Changes in Object-Oriented Software Systems	IEEE	(Sharafat; Tahvildari, 2007)
25	2010	A Replicated and Refined Empirical Study of the Use of Friends in C++ Software	Scopus	(Englishet al., 2010)
26	2010	A Quantitative Approach to Software Maintainability Prediction	IEEE	(Ping, 2010)
27	2007	Assessing Software System Maintainability using Structural Measures and Expert Assessments	IEEE	(Anda, 2007)

"Tabela 3, continuação"

#	Ano	Título	Fonte	Ref.
28	2010	Assessing Traditional and New Metrics for Object-Oriented Systems	Scopus	(Concas et al., 2010)
29	2007	Assessment of Package Cohesion and Coupling Principles for Predicting the Quality of Object Oriented Design	IEEE	(Atole; Kale, 2006)
30	2010	A Study of the Relationships between Source Code Metrics and Attractiveness in Free Software Projects	IEEE	(Meirelles et al., 2010)
31	2005	A UML Approximation of Three Chidamber-Kemerer Metrics and Their Ability to Predict Faulty Code Across Software Projects	IEEE	(Cruz; Ochimizu, 2010)
32	2006	Building Scalable Failure-proneness Models Using Complexity Metrics for Large Scale Software Systems	IEEE	(Bhat; Nagappan, 2010)
33	2010	Can Complexity, Coupling, and Cohesion Metrics be Used as Early Indicators of Vulnerabilities?	Scopus	(Chowdhury; Zulkernine, 2010)
34	2008	Change Prediction in Object-Oriented Software Systems: A Probabilistic Approach	Scopus	(Sharafat; Tahvildari 2008)
35	2011	Design Evolution Metrics for Defect Prediction in Object Oriented Systems	Springer	(Kpodjedo et al., 2011)
36	2006	*Dn*-Based Design Quality Comparison of Industrial Java Applications	IEEE	(Roubtsov et al., 2009
37	2006	Empirical Analysis of Object-Oriented Design Metrics for Predicting High and Low Severity Faults	Scopus	(Zhou; Leung, 2006)
38	2007	Empirical Analysis of Software Fault Content and Fault Proneness Using Bayesian Methods	Scopus	(Pai; Dugan, 2007)
39	2012	Empirical Investigation of Fault Prediction Capability of Object Oriented Metrics of Open Source Software	IEEE	(Singh; Verma, 2012)
40	2006	Empirical Study of Object-Oriented Metrics	Scopus	(Aggarwal et al., 2006)
41	2009	Empirical Validation of Object-Oriented Metrics for Predicting Fault Proneness Models	Scopus	(Singh et al., 2010)
42	2005	Empirical Validation of Object-Oriented Metrics on Open Source Software for Fault Prediction	Scopus	(Gyimothy et al., 2005)

"Tabela 3, continuação"

#	Ano	Título	Fonte	Ref.
43	2007	Empirical Validation of Three Software Metrics Suites to Predict Fault-Proneness of Object-Oriented Classes Developed Using Highly Iterative or Agile Software Development Processes	IEEE	(Olague et al., 2007)
44	2012	Evaluating the Correlation Between Software Defect and Design Coupling Metrics	IEEE	(Abuasad; Alsmadi, 2012)
45	2009	Examining the Potentially Confounding Effect of Class Size on the Associations Between Object-Oriented Metrics and Change-Proneness	Scopus	(Zhouet al., 2009)
46	2009	Fault Detection and Prediction in an Open-Source Software Project	Scopus	(English et al., 2009)
47	2012	Fault Prediction and the Discriminative Powers of Connectivity-Based Object-Oriented Class Cohesion Metrics	Scopus	(Al Dallal, 2012)
48	2009	Finding Software Metrics Threshold Values Using ROC Curves	Scopus	(Shatnawi et al., 2010)
49	2006	Identification of Defect-Prone Classes in Telecommunication Software Systems Using Design Metrics	Scopus	(Janes et al., 2006)
50	2009	Identifying Thresholds for Object-Oriented Software Metrics	Science	(Ferreira et al., 2012)
51	2011	Impact of Attribute Selection on Defect Proneness Prediction in OO Software	Scopus	(Mishra; Shukla, 2011)
52	2011	Improving the Applicability of Object-Oriented Class Cohesion Metrics	Scopus	(Al Dallal, 2011a)
53	2012	Investigating Object-Oriented Design Metrics to Predict Fault-Proneness of Software Modules	IEEE	(Rathore; Gupta, 2012)
54	2013	Investigation of Domain Effects on Software	Scopus	(Virani et al., 2009)
55	2013	Investigation of Relationship Between Object-Oriented Metrics and Change Proneness	Compendex	(Malhotra; Khanna, 2013)
56	2009	Is Depth of Inheritance Tree a Good Cost Prediction for Branch Coverage Testing?	IEEE	(Shaheen; Bousquet, 2009)
57	2011	Measuring the Discriminative Power of Object-Oriented Class Cohesion Metrics	IEEE	(Al Dallal, 2011b)
58	2005	Measuring the Impact of Friends on the Internal Attributes of Software Systems	IEEE	(English et al., 2005)
59	2007	Metrics and Evolution in Open Source Software	IEEE	(Leeet al., 2007)

"Tabela 3, continuação"

#	Ano	Título	Fonte	Ref.
60	2013	Object-Oriented Class Maintainability Prediction Using Internal Quality Attributes	Compendex	(Al Dallal, 2013)
61	2007	Object-Oriented Software Fault Prediction Using Neural Networks	Compendex	(Kanmani et al., (2007)
62	2004	Predicting Class Testability Using Object-Oriented Metrics	IEEE	(Bruntink; Deursen, 2004)
63	2009	Predicting Code Change by Using Static Metrics	IEEE	(Mauczka et al., 2009)
64	2006	Predicting Fault-Prone Components in a Java Legacy System	Compendex	(Arisholm; Briand, 2006)
65	2007	Predicting Object-Oriented Software Maintainability Using Multivariate Adaptive Regression Splines	Scopus	(Zhou; Leung, 2007)
66	2008	Predicting the Maintainability of Open Source Software Using Design Metrics	Springer	(Zhou; Xu, 2008)
67	2009	Quantitative Evaluation of Software Quality Metrics in Open-Source Projects	IEEE	(Barkmann et al., 2010)
68	2010	Size, Inheritance, Change and Fault-Proneness in C# Software	Scopus	(Gatrell; Counsell, 2010)
69	2013	Statistical Comparison of Modelling Methods for Software Maintainability Prediction	Scopus	(Kaur; Kaur, 2013)
70	2012	The Ability of Object-Oriented Metrics to Predict Change-Proneness: A Meta-Analysis	Scopus	(Luet al., 2012)
71	2012	The Impact of Accounting for Special Methods in the Measurement of Object-Oriented Class Cohesion on Refactoring and Fault Prediction Activities	Scopus	(Al Dallal, 2012)
72	2006	The Interpretation and Utility of Three Cohesion Metrics for Object-Oriented Design	Scopus	(Counsell et al., 2006)
73	2010	Towards Estimating Physical Properties of Embedded Systems using Software Quality Metrics	IEEE	(Corrêa et al., 2010)
74	2011	Transitive-Based Object-Oriented Lack-of-Cohesion Metric	Science	(Al Dallal, 2011c)
75	2011	Using a Class Abstraction Technique to Predict Faults in OO Classes: A Case Study Through Six Releases of the Eclipse JDT	Scopus	(Babich et al., 2011)
76	2011	Using Complexity, Coupling, and Cohesion Metrics as Early Indicators of Vulnerabilities	Scopus	(Chowdhury; Zulkernine, 2011)
77	2011	Using Source Code Metrics to Predict Change-Prone Java Interfaces	IEEE	(Romano; Pinzger, 2011)

"Tabela 3, conclusão"

#	Ano	Título	Fonte	Ref.
78	2012	Validating the Effectiveness of Object-Oriented Metrics over Multiple Releases for Predicting Fault Proneness	IEEE	(Rathore; Gupta, 2012)
79	2010	Validation of CK Metrics for Object Oriented Design Measurement	IEEE	(Kulkarni et al., 2010)

5.4.1 Análise quantitativa - questões de pesquisa

Análise quantitativa dos resultados das duas questões de pesquisa elaboradas é discutida neste item. Considerando a questão

Q1: Quais são as principais medidas investigadas, na última década, para avaliar a qualidade interna de sistemas de software orientados a objeto?

Foram selecionados artigos que utilizaram medidas de sistemas de software desenvolvidos com tecnologias tradicionais e OO para quantificar a qualidade interna. Ao final, foram registradas 265 medidas. Utilizando o critério de citação em 10 ou mais artigos, foram identificadas 15 principais medidas associadas a cinco propriedades diferentes. Essas medidas são apresentadas na Tabela 4, juntamente com as propriedades com as quais foram relacionadas, a quantidade de citações e em quais estudos elas foram encontradas.

Tabela 4 Principais medidas utilizadas para avaliar qualidade interna de sistemas de software OO

#	Medida	Propriedade	Quantidade de Citações
1	*Lack of Cohesion Between Objects (LCOM)*	Coesão	**40**
2	*Depth of Inheritance Tree (DIT)*	Herança	**37**
3	*Response For Class (RFC)*	Acoplamento	**33**
4	*Coupling Between Objects (CBO)*	Acoplamento	**32**
5	*Number of Children (NOC)*	Herança	**30**
6	*Weight Methods per class (WMC)*	Complexidade	**28**
7	*Lines of Code (LOC)*	Tamanho	**20**
8	*Number of Methods (NOM)*	Tamanho	**18**
9	*Cyclomatic Complexity (CC)*	Complexidade	**13**
10	*Number of Attributes (NOA)*	Tamanho	**11**
11	*Tight Class Cohesion (TCC)*	Coesão	**10**
12	*Lack of Cohesion Between Objects (LCOM4)*	Coesão	**10**
13	*Fan-Out*	Acoplamento	**10**
14	*Loose Class Cohesion (LCC)*	Coesão	**10**
15	*Lack of Cohesion Between Objects (LCOM2)*	Coesão	**10**

Apesar da quantidade de medidas identificadas, muitas são consideradas variações de medidas bem estabelecidas, por exemplo, as medidas CBO', CBO_IUB e CBO_U são variações da medida CBO (AL DALLAL, 2013). Outras medidas foram citadas em apenas um trabalho, pois caracterizam um aspecto específico da qualidade, por exemplo, a medida IUC utilizada para quantificar a coesão externa de classes do tipo interface em Java (MEIRELLES et al., 2010).

Em relação ao modelo de qualidade do produto da ISO/IEC 25010 (INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 2011), pode-se verificar que essas medidas foram utilizadas para avaliar a qualidade interna de sistemas de software para as características Adequação Funcional, Usabilidade, Confiabilidade, Segurança e Manutenibilidade. Em 77 artigos, as medidas registradas foram utilizadas para verificar se são bons indicadores de qualidade em relação à Manutenibilidade (Tabela 5).

Tabela 5 Características de qualidade da ISO/IEC 25010 abordadas nos trabalhos selecionados

#	Característica	Quantidade de Citações
1	Manutenibilidade	**77**
2	Adequação Funcional	**4**
3	Segurança	**2**
4	Usabilidade	**1**
5	Confiabilidade	**1**
6	Compatibilidade	**0**
7	Eficiência no Desempenho	**0**
8	Portabilidade	**0**

Em relação à característica **Segurança**, foi verificado em dois artigos, se medidas de software são capazes de identificar vulnerabilidades no sistema (CHOWDHURY; ZULKERNINE, 2010; CHOWDHURY; ZULKERNINE, 2011). A característica **Adequação Funcional** foi abordada em artigos que buscavam identificar propensão a falhas e propensão a mudanças (AL DALLAL; BRIAND, 2010; CRUZ; OCHIMIZU, 2010; PIZZI, 2013; XIAO, 2013). Em relação à característica **Manutenibilidade**, as medidas foram definidas/utilizadas nos contextos de:

a) Estimar a propensão a falhas em classes OO, a propensão a mudanças e a propensão a erros;
b) Reduzir o custo de testes e o grau de conectividade entre as classes;
c) Identificar classes que violam princípios de projeto;
d) Predizer a manutenibilidade em classes, a evolução de projetos *open-source*, a variabilidade das medidas e o impacto da manutenibilidade em sistemas de software considerando domínios diferentes.

Em geral, para avaliar a ocorrência de tais eventos, os artigos selecionados utilizaram técnicas/modelos estatísticos. Esses artigos e o contexto da utilização dessas técnicas são abordados na questão

Q2: Quais são os principais técnicas/modelos estatísticos utilizados, na última década, por pesquisadores da área Engenharia de Software, para avaliar a qualidade interna de sistemas de software orientados a objeto?

Os artigos selecionados possuem metodologias e objetivos diversos. Ao final da execução da RSL, foram registradas 40 técnicas diferentes, sendo as 10 mais citadas apresentadas na Tabela 6. Muitos estudos utilizaram mais de uma técnica estatística para avaliar os resultados sobre a qualidade interna de sistemas de software. A escolha de uma técnica pode ser dependente dos objetivos da pesquisa.

Tabela 6 Principais técnicas/modelos estatísticos utilizados para avaliar qualidade interna de sistemas de software OO

#	Técnica estatística	Quantidade de Citações
1	Análise Descritiva	**42**
2	Testes de Correlação	**39**
3	Regressão Logística	**36**
4	Redes Neurais Artificiais	**13**
5	Análise de Componentes Principais	**7**
6	Probabilidade	**3**
7	Análise de Colinearidade	**3**
8	Testes Não Paramétricos	**3**
9	Inferência Estatística	**3**
10	Metanálise	**2**

Muitos estudos utilizaram mais de uma técnica estatística para avaliar os resultados sobre a qualidade interna de sistemas de software. A escolha de uma técnica pode ser dependente dos objetivos da pesquisa. A técnica mais utilizada

foi a **Análise Descritiva** (42 artigos - 85,7%), que visa descrever e sumarizar informações de uma população, como os valores médios, os máximos, os mínimos, a variância e o desvio padrão.

Em seguida, os **Testes de Correlação** foram abordados em 39 artigos (79,6%). O objetivo da utilização desses testes foi verificar a relação entre as medidas de software, por exemplo, se os valores obtidos em medidas de complexidade estão relacionados a valores de medidas de tamanho (quanto maior o software mais complexo ele pode ser). Os tipos de teste mais utilizados foram o **Teste de Spearman** (16 artigos - 32,6%), o **Teste de Pearson** (14 artigos - 28,6%) e o **Teste de Kendall** (dois artigos - 4,1%). O tipo de teste de correlação aplicado na amostra em estudo não foi especificado em cinco artigos.

A terceira técnica mais utilizada pelos pesquisadores foi a **Regressão Logística** (univariada e multivariada). O contexto da utilização da técnica é a criação de modelos de predição de falhas e de mudanças em sistemas de software. Em alguns estudos, apenas modelos baseados em regressão univariada (um modelo para cada medida considerada como variável) são propostos. Os modelos de predição de falhas e de predição de mudanças também são propostos utilizando as **Redes Neurais Artificiais** com algoritmos de aprendizagem de máquina. A **Análise de Componentes Principais** (ACP) foi mencionada em sete estudos, cujo objetivo é reduzir a aplicabilidade de medidas e identificar as dimensões pelas quais essas medidas juntas podem explicar características de sistemas. Em alguns trabalhos, a ACP foi utilizada para identificar medidas mais relevantes para o objetivo da pesquisa e, em seguida, a correlação entre elas era avaliada utilizando os **Testes de Correlação** mencionados anteriormente (ARISHOLM; BRIAND, 2006; AL DALLAL, 2012; BADRI; BADRI; TOURÉ, 2011; BHAT; NAGAPPAN, 2006; ETZKORN et al., 2004; KANMANI et al., 2007; OLAGUE et al., 2008).

Por fim, alguns trabalhos utilizaram os **Testes de Probabilidade** (FERREIRA et al., 2012; SHARAFAT; TAHVILDARI, 2007; SHARAFAT; TAHVILDARI, 2008), a **Análise de Colinearidade** (BABICH et al., 2011; OLAGUE et al., 2007; SUCCI et al., 2005), os **Testes Não Paramétricos** (por exemplo, o **Teste de Wilcoxon**) (CHOWDHURY; ZULKERNINE, 2010; SHAHEEN; DU BOUSQUET, 2009; ZHOU; LEUNG, 2007), as **Técnicas de Inferência Estatísticas** (por exemplo, o Teste Chi-Quadrado) (PARTHASARATHY; ANBAZHAGAN, 2006), ***fuzzy*** (PIZZI, 2013; XU; HO; CAPRETZ, 2008) e a **Metanálise** (LU et al., 2012). Outras técnicas foram abordadas apenas uma vez, por exemplo, a **Análise Discriminante** (BRUNTINK; DEURSEN, 2004), a técnica **ANOVA** (VIRANI et al., 2009), a **Análise ROC** (SHATNAWI et al., 2010), a técnica **MARS** (ZHOU; XU, 2008), a técnica de ***Confounding Effect*** (ZHOU; LEUNG; XU, 2009), o **Modelo Oculto de Markov** (KAYARVIZHY; KANMANI, 2011) e os **Modelos Bayesianos** (PAI; DUGAN, 2007).

5.4.2 Análise qualitativa - resultados gerais e pesquisa futura

Os resultados da RSL sugerem que não há escolhas óbvias em relação a quais medidas e quais técnicas os pesquisadores devem utilizar. Os métodos estatísticos são escolhidos de acordo com os objetivos da pesquisa; por isso, mostrar, predizer ou otimizar são obtidos por diferentes métodos. Os 79 artigos obtiveram resultados diferentes em relação às melhores medidas e o resultado estatístico. Cabe ressaltar que alguns artigos selecionados utilizaram bases de dados de medidas existentes, como a base da NASA (ZHOU; LEUNG, 2006; XU; HO; CAPRETZ, 2008) e a base PROMISE (RATHORE; GUPTA, 2012), e aqueles que montaram suas próprias bases, coletando o valor das medidas nos

sistemas selecionados, não as disponibilizaram em meios de comunicação científica.

Em relação as medidas, a suíte CK foi a mais utilizada e citada, conforme os resultados obtidos na análise quantitativa da Q1. Vários artigos apontam essas medidas, ainda que pioneiras e, portanto, justificam o seu amplo uso em artigos da última década, como boas preditoras da qualidade de código OO. Além disso, um artigo propõe a utilização das medidas na avaliação de sistemas de software por meio de modelos em UML (*Unified Modeling Language*) (CRUZ; OCHIMIZU, 2010) e compara os resultados da avaliação com o valor das medidas coletadas no código dos sistemas. O resultado dessas medidas foi igualmente bom aos coletados no código dos sistemas, comprovando evidências de que elas são significativas para a qualidade de software.

Das técnicas/modelos estatísticos, a maioria dos trabalhos utiliza a **Análise Descritiva** para caracterizar sistemas de software, por exemplo, **CBO** e **LCOM**, com valor médio alto e baixo, respectivamente, podem indicar um sistema com alto acoplamento e baixa coesão (AL DALLAL, 2013). Entretanto, a **Análise Descritiva** fornece resumos simples sobre a amostra e as observações feitas, sendo que não há conclusão sobre a relação entre as variáveis em estudo. Com técnicas de **Análise Multivariada**, podem ser feitas inferências sobre o relacionamento entre variáveis diferentes, que incluem técnicas de dependência (a **Regressão Múltipla**, a **Análise Discriminante**, a **Análise Multivariada de Variância - MANOVA**) e técnicas de interdependência (os **Testes de Correlação**, a **Covariância**, a **Análise de Agrupamento** ou ***Cluster***, a **Análise Fatorial**, a **Análise de Correspondência**).

Dos modelos propostos na literatura, há iniciativas que contribuem com a identificação de dimensões, com a classificação de características, com a classificação da confiabilidade interna e com a identificação de medidas que

preveem características de qualidade (ANDA, 2007; KAYARVIZHY; KANMANI, 2011). Entretanto, elas não abrangem medidas de propriedades como a complexidade, a herança, o polimorfismo, o encapsulamento, o acoplamento, a coesão e o tamanho. Portanto, não se sabe se, em um contexto de muitas variáveis, as medidas escolhidas pelos autores são realmente relevantes para determinar a qualidade de sistemas. Além disso, os modelos de predição focam em identificar medidas que podem ser boas preditoras de qualidade, sob o aspecto da manutenibilidade (ABUASAD; ALSMADI, 2012; AGGARWAL et al., 2006; AL DALLAL, 2012a; ATOLE; KALE, 2006; BADRI; BADRI; TOURÉ, 2011; BHAT; NAGAPPAN, 2006; ENGLISH et al., 2009; ESKI; BUZLUCA, 2011; GYIMOTHY; FERENC; SIKET, 2005; KIEWKANYA; JINDASAWAT; MUENCHAISRI, 2004; KPODJEDO et al., 2011; LEUNG; XU, 2009; OLAGUE et al., 2007; OLAGUE et al., 2008; PAI; DUGAN, 2007; ROUBTSOV; SEREBRENIK; VAN DEN BRAND, 2009; SINGH; VERNA, 2012; ZHOU; LEUNG, 2006; ZHOU et al., 2010). Em dois artigos, cujo foco é identificar medidas de software indicadoras de vulnerabilidades, os autores não chegam aos méritos da criação de modelos (ROMANO; PINZGER, 2011; SHARAFAT; TAHVILDARI, 2008). Dentre os selecionados, não foram encontrados artigos que tinha como objetivo avaliar a similaridade entre sistemas com base na qualidade interna, ou seja, a criação de um modelo de qualidade baseado na **Análise de Agrupamento** (*cluster*).

Em relação as linguagens, a mais utilizada foi Java (55%) e C++ foi a segunda mais abordada (21%). Os sistemas de software nas linguagens C (7%), C# (3%), Python (2%), PHP (1%), Perl (1%) e Ada (3%) também serviram como amostra para pesquisas. A linguagem utilizada não foi declarada em sete artigos. Alguns artigos justificam a preferência por sistemas desenvolvidos em Java: popularidade na OO (AL DALLAL, 2012; AL DALLAL; BRIAND, 2010; FAROOQ; BRAUNGARTEN; DUMKE, 2005).

Cabe ressaltar que seis artigos tiveram, como amostra, sistemas de software proprietários, outros artigos utilizaram sistemas *open-source*:

a) A Large-Scale Empirical Comparison of Object-Oriented Cohesion Metrics (12 sistemas desenvolvidos em Java cuja empresa e sua localização não puderam ser reveladas) (BARKER; TEMPERO, 2007);
b) Assessing Software System Maintainability using Structural Measures and Expert Assessments (quatro sistemas de companhias norueguesas) (ANDA, 2007);
c) Building Scalable Failure-proneness Models Using Complexity Metrics for Large Scale Software Systems (duas versões do Windows) (BHAT; NAGAPPAN, 2006);
d) Dn-Based Design Quality Comparison of Industrial Java Applications (dois sistemas financeiros de uma empresa irlandesa) (ROUBTSOV; SEREBRENIK; VAN DEN BRAND, 2009);
e) Identification of Defect-Prone Classes in Telecommunication Software Systems using Design Metrics (cinco sistemas de uma empresa norte-americana) (JANES et al., 2006);
f) Predicting Class Testability using Object-Oriented Metrics (um sistema desenvolvido pelo *Software Improvement Group* – SIG) (BRUNTINK; DEURSEN, 2004).

A opção pela utilização de sistemas *open-source* pode ser explicada, pois, além da facilidade de obtê-los em repositórios *on-line*, seu uso tem aumentado significativamente, a ponto de tornar-se influente para a economia global. No artigo "A Study of the Relationships between Source Code Metrics and Attractiveness in Free Software Projects" (MEIRELLES et al., 2010),

publicado nos Anais do Simpósio Brasileiro de Engenharia de Software (SBES) de 2010, é ressaltada a satisfação dos usuários e a qualidade em sistemas *open-source*, são alcançadas por causa da colaboração da comunidade de usuários e desenvolvedores, que relata falhas, corrige *bugs* e adiciona recursos de maneira rápida e organizada.

Outro ponto a ser observado é a variabilidade da população estudada. Aproximadamente, 17 artigos consideram que o domínio no qual o software pertence é parte fundamental para a generalização dos resultados. Em seis estudos (AL DALLAL; BRIAND, 2010; ENGLISH et al., 2010; ETZKORN et al., 2004; FERREIRA et al., 2012; KANMANI et al., 2007; RATHORE; GUPTA, 2012), os autores afirmaram que o resultado das medidas pode variar em sistemas de domínios diferentes por possuírem alternativas de *design* diferentes. Esses dados sugerem um campo aberto para estudos, principalmente empíricos, que objetivam:

a) Avaliar a qualidade interna de sistemas de software, considerando os domínios em que estão inseridos;
b) Avaliar a qualidade interna de sistemas sob a característica Manutenibilidade, considerando a amplitude das medidas, as suas propriedades e a sua relevância;
c) Avaliar a qualidade interna sob outras características definidas na ISO/IEC 25010;
d) Mapear e sintetizar essas características em modelos para determinar se as medidas são capazes de predizer e caracterizar a qualidade de sistemas de software OO, visto que os modelos propostos não são generalizáveis e não abrangem todas as medidas e as propriedades existentes;

e) Mensurar e estimar valores de referência para domínios diferentes, considerando medidas diferentes;
f) Verificar similaridades entre características de sistemas de domínios diferentes.

5.5 Revisão de literatura - outros trabalhos

Inicialmente, uma revisão de literatura foi executada para buscar informações sobre o tema proposto. Essa primeira revisão, executada de maneira informal, corroborou para o ajuste do tema e da investigação do estado da arte da qualidade interna de sistemas de software, formalizada pela RSL. Nesta seção, são citados dois trabalhos relevantes e relacionados ao tema de pesquisa, em ordem cronológica.

No primeiro trabalho (BRIAND et al., 2000), os autores propuseram a investigação de como medidas de software OO capturam aspectos estruturais de projetos para determinar a qualidade desses projetos e com que precisão essas medidas podem prever a propensão a falhas em classes, utilizando as técnicas estatísticas **Análise de Componentes Principais (ACP)** e a **Regressão Logística**, respectivamente. O resultado foi um modelo de predição de falhas em classes com porcentagem de cerca de 80% de classificações corretas, classificando mais de 90% de classes com falhas. Entretanto, o estudo foi realizado em oito sistemas de software desenvolvidos por estudantes da University of Maryland, cuja experiência era limitada.

Em outro trabalho (SOUZA; MAIA, 2013), são propostos valores de referência para um conjunto de medidas de acoplamento. Foram coletados 100 sistemas de software no repositório *open-source* Sourcerer[8]. Essa amostra

[8] http://sourcerer.ics.uci.edu/

continha sistemas de diferentes categorias ou domínios. Os resultados mostraram que o uso de valores de referência genéricos, para a comparação de sistemas de software pertencentes a diferentes categorias, pode não ser adequado. Isso acontece, pois, um valor de referência, que detectar determinadas características em uma categoria, por exemplo, a alta dependência entre os tipos (acoplamento), pode não detectar o mesmo grau de dependência em outra categoria.

5.6 Ameaças à validade

No estudo realizado, podem ser identificadas algumas ameaças à validade, por exemplo:

a) Quanto à validade do construto:

- A elaboração da *string* de busca procurou ser o mais abrangente possível para capturar trabalhos considerados mais relevantes para o tema de pesquisa. Entretanto, mesmo que a *string* de busca seja devidamente derivada das questões de pesquisa, é possível que resultados relevantes tenham sido excluídos nas combinações dos termos inseridos;
- É possível que a exclusão de trabalhos relacionados ao tema, em casos em que trabalhos não possuam título, resumo e palavras-chave bem definidas e alinhadas ao tema em estudo, tenha limitado o número de trabalhos que podem acrescer informações interessantes à pesquisa.

b) Quanto à validade interna e externa:

- A fim de minimizar qualquer ameaça à validade interna, principalmente na seleção de estudos primários e extração de

dados, os pesquisadores envolvidos na RSL conduziram suas atividades paralelamente e qualquer discordância foi discutida até chegar a um consenso;

- Não foram identificados fatores de ameaça externa.

5.7 Considerações finais

Nesta seção, foram apresentados resultados de uma RSL, cujo interesse de pesquisa foi identificar estudos sobre a avaliação da qualidade interna de sistemas de software OO, utilizando técnicas/modelos estatísticos. Como resultado, foi obtido o total de 8.231 trabalhos, dos quais 79 trabalhos (estudos) foram selecionados como estudos primários. Na Fase Análise dos Resultados, foram identificadas 265 medidas utilizadas para quantificar atributos de qualidade em sistemas de software, porém apenas 15 (as mais citadas), foram consideradas principais para responder a primeira questão de pesquisa.

Em relação às técnicas/modelos estatísticas, foram identificadas 40 técnicas/modelos, sendo que foram apresentadas apenas as 10 principais. O resultado da RSL sugere a existência de um conjunto extenso de medidas relacionadas, mas, muitas vezes, podem representar uma mesma característica, ou seja, derivações e adaptações similares. Além disso, os estudos primários selecionados não consideram todas as propriedades existentes para caracterizar a qualidade de um sistema de software. Muitos desses estudos focam na complexidade, na coesão e no tamanho. Dentre as contribuições do estudo feito nesta seção, podem ser listadas:

a) A identificação de um conjunto das principais medidas utilizadas para quantificar a qualidade interna de sistemas de software OO;

b) A identificação das principais técnicas/modelos estatísticos utilizados para a avaliação da qualidade de software;
c) Identificação das principais referências bibliográficas para o tema em pesquisa.

Dessa forma, com os fundamentos teóricos discutidos, estudos empíricos devem ser realizados para avaliar a qualidade interna de sistemas de software, a fim de elaborar modelos que possam caracterizar a qualidade, considerando os domínios de software, as medidas relevantes para quantificá-la e suas propriedades. Além disso, uma base de dados científica de medidas de software deve ser elaborada para outros pesquisadores poderem utilizá-la em estudos subsequentes.

6 *O3SMEASURES* - UM APOIO COMPUTACIONAL PARA EXTRAÇÃO DE VALOR DE MEDIDAS DE SISTEMAS DE SOFTWARE ORIENTADOS A OBJETOS

6.1 Considerações iniciais

A medição do código por ferramentas computacionais é frequentemente utilizada para avaliar a qualidade interna de um sistema de software. Neste trabalho, foi desenvolvido um apoio computacional (*plug-in* para o Eclipse IDE), cujo objetivo é avaliar sistemas, utilizando as medidas identificadas na RSL previamente realizada, para caracterizar a qualidade desses sistemas. O desenvolvimento desse apoio computacional é justificado pela ausência de uma ferramenta automatizada para medir sistemas de software OO desenvolvido em Java, que possua em seu catálogo de medidas, por exemplo, as medidas **LCOM2**, **LCOM4** e **LCC**, consideradas revelantes para avaliar a qualidade interna desses sistemas, conforme apresentado na RSL (Seção 5).

Esta seção está organizada da seguinte forma. A descrição do problema é brevemente apresentada na subseção 6.2. A modelagem do *plug-in O3SMeasures* é apresentada na subseção 6.3. A descrição das medidas implementadas no *plug-in* é apresentada na subseção 6.4. O funcionamento do *plug-in* é descrito na subseção 6.5.

6.2 Descrição do problema

Ferramentas computacionais para medir o código são frequentemente utilizadas para avaliar a qualidade de um sistema de software e suas propriedades. Ao longo dos anos, foram desenvolvidas ferramentas, por

exemplo, os *plug-ins* Metrics[9] e VizzMaintenance[10] para o Eclipse IDE para medir sistemas de software OO. Entretanto, não foram encontradas ferramentas que tivessem em sua composição, medidas consideradas pelos pesquisadores na RSL como as principais utilizadas para avaliar a qualidade de sistemas desenvolvidos em Java.

Dessa forma, foi desenvolvido um apoio computacional chamado *O3SMeasures* (*Object-Oriented Open-Source Software Measures*) como um *plug-in* para o Eclipse IDE para medir código de sistemas de software OO desenvolvidos em Java. Para desenvolvê-lo, foram utilizados o Eclipse IDE 4.4 (Luna), *Java Development Tools* (JDT), *Plug-in Development Environment* (PDE) e *Abstract Syntax Tree* (AST). No JDT, há ferramentas para manipular código Java. No PDE, há ferramentas para desenvolver e testar *plug-ins* no Eclipse IDE.

O *plug-in* AST é a estrutura base para ferramentas do Eclipse IDE, incluindo *Refactoring*, *QuickFix* e *QuickAssist*, cuja função é mapear código Java em forma de representação de uma árvore. Essa representação torna a análise do código mais simples e confiável, além de facilitar a modificação programática do código baseado em texto. Como ferramenta de teste da funcionalidade do *plug-in* desenvolvido, Junit4 foi utilizado, *framework* para escrever testes repetíveis em Java. Em resumo, *O3SMeasures* depende dos pacotes:

a) org.eclipse.core.resources;
b) org.eclipse.jdt;
c) org.eclipse.jdt.core (3.8.3);
d) org.eclipse.core.runtime (3.8.0);

[9] http://metrics.sourceforge.net/
[10] http://www.arisa.se/products.php

e) org.eclipse.ui;
f) org.eclipse.ui.ide;
g) org.junit (4.11.0).

6.3 Modelagem

No *plug-in O3SMeasures*, existem 16 medidas implementadas e organizadas conforme Tabela 7:

a) **Propriedade**. Trata-se de uma abstração que caracteriza um objeto, por exemplo, se a medida tem por objetivo mensurar o tamanho do software (em termos de manutenção ou de esforço) ou o quão coeso é o software. Há cinco propriedades: i) tamanho; ii) herança; iii) complexidade; iv) coesão; e v) acoplamento;
b) **Granularidade**. Define sob qual nível ou extensão as medidas são aplicadas. No *O3SMeasures*, as medidas podem ser aplicadas em três níveis: i) projeto; ii) pacote; e iii) classe.

Tabela 7 Medidas implementadas no *plug-in O3SMeasures*

#	Medida	Propriedade	Granularidade
1	*Lines of Code (LOC)*	Tamanho	Projeto
2	*Number of Classes (NC)*	Tamanho	Projeto
3	*Number of Attributes (NOA)*	Tamanho	Projeto
4	*Number of Methods (NOM)*	Tamanho	Projeto
5	*Response for a Class (RFC)*	Acoplamento	Classe
6	*Coupling Between Objects (CBO)*	Acoplamento	Classe
7	*Fan-out*	Acoplamento	Pacote
8	*Weight Methods per Class (WMC)*	Complexidade	Classe
9	*Cyclomatic Complexity (CC)*	Complexidade	Classe
10	*Depth of Inheritance Tree (DIT)*	Herança	Pacote
11	*Number of Children (NOC)*	Herança	Classe

"Tabela 7, conclusão"

#	Medida	Propriedade	Granularidade
12	*Lack of Cohesion Between Methods (LCOM)*	Coesão	Classe
13	*Lack of Cohesion Between Methods (LCOM2)*	Coesão	Classe
14	*Lack of Cohesion Between Methods (LCOM4)*	Coesão	Classe
15	*Loose Class Cohesion (LCC)*	Coesão	Classe
16	*Tight Class Cohesion (TCC)*	Coesão	Classe

Com a definição das medidas e das ferramentas utilizadas, a estrutura do *plug-in O3SMeasures* foi modelada. O modelo com as entidades e os relacionamentos propostos para a implementação, é ilustrado no Diagrama de Classes apresentado na Figura 8. As classes são:

a) ***Application***: classe principal do *plug-in O3SMeasures*, na qual as medidas utilizadas são instanciadas e o método de execução da medição é invocado;
b) ***ItemMeasured***: representa o item medido;
c) ***Measure***: representa as medidas de software;

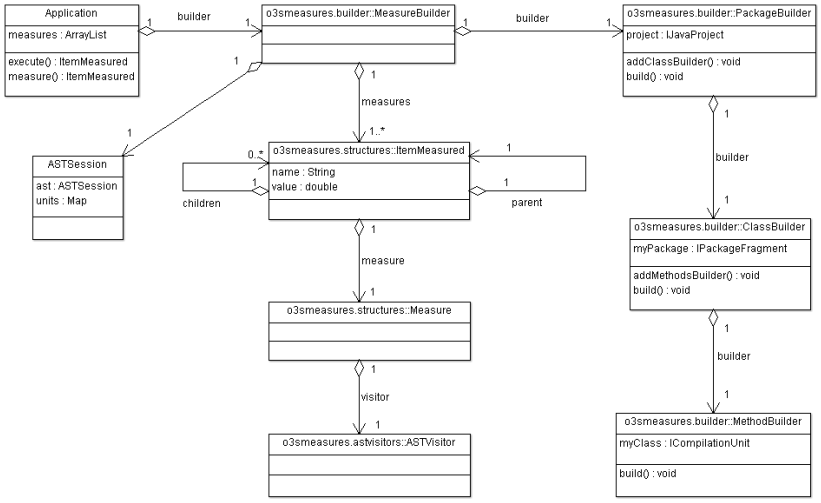


Figura 8 Diagrama de classes do *plug-in O3SMeasures*

d) ***MeasureBuilder***: implementação da interface *IBuilder* para medidas. Nessa classe, é verificada a granularidade da medida recebida por parâmetro, redirecionando a avaliação para o ***PackageBuilder*** adequado;
e) ***PackageBuilder***: implementação da interface *IBuilder* para avaliação e apresentação dos resultados da medição, considerando pacotes;
f) ***ClassBuilder***: implementação da interface *IBuilder* para avaliação e apresentação dos resultados da medição, considerando classes;
g) ***MethodBuilder***: implementação da interface *IBuilder* para avaliação e apresentação dos resultados da medição, considerando métodos.

O mapeamento da estrutura do código do sistema de software foi feito utilizando classes do *plug-in* AST e o *Java Model* do *plug-in* JDT. Cada sistema

de software OO Java é representado internamente no Eclipse como um modelo Java (*Java Model*). Esse modelo não contém informações detalhadas como a AST, mas é simples e prático de criar. Por exemplo, a *view* Outline do Eclipse IDE utiliza o modelo Java para sua representação; dessa forma, as informações da *view* podem ser rapidamente atualizadas. Entretanto, o modelo Java é representado como uma estrutura de árvore que pode ser descrita por meio dos elementos listados na Tabela 8.

Tabela 8 Estrutura do modelo Java

Elemento do Projeto	**Elemento do *Java Model***	**Descrição**
Projeto desenvolvido em Java	*IJavaProject*	O projeto Java com todos os outros objetos.
Pastas *src/bin* ou bibliotecas externas	*IPackageFragmentRoot*	Pastas ou bibliotecas (*zip/jar*) que mantém os arquivos binários ou fonte.
Pacotes	*IPackageFragment*	Cada pacote está abaixo do *IPackageFragmentRoot* e sub-pacotes não 'nós-folha' do pacote são listados diretamente sob *IPackageFragmentRoot*.
Código Java	*ICompilationUnit*	O código está sempre abaixo do nó pacote.
Tipos/Atributos/Métodos	*IType/IField/IMethod*	Tipos, atributos e métodos em uma classe.

A AST é uma representação em árvore detalhada do código Java descrito na Tabela 8 e define uma API (*Application Programming Interface*) para modificar, criar, ler e apagar o código. Quando o *plug-in* é iniciado, instanciando a classe Application, uma ASTSession é criada para armazenar a estrutura do projeto alvo da medição. Cada arquivo fonte Java é representado como uma subclasse da classe ASTNode. Cada nó específico da AST fornece informações específicas sobre o objeto que ele representa. Por exemplo, o

método methodDeclaration() é utilizado para obter a assinatura dos métodos declarados na classe e o método variableDeclarationFragment() é utilizado para obter as variáveis declaradas na classe. A AST é criada com base em uma ICompilationUnit do *Java Model*, cuja organização é traduzida em uma estrutura de árvore AST pela classe ASTParser.

Esse resultado pode ser acessado, pois o código distribuído na AST pode ser visitado utilizando a estrutura ASTVisitor. No *O3SMeasures*, as medidas possuem uma classe do tipo visitante (Visitor), que extende uma ASTVisitor, em que a lógica do cálculo dessas medidas é implementada e mostrada nas *views* do *plug-in*. Um exemplo do código do *plug-in* pode ser visualizado na Figura 9, em que se observa que a classe *MeasureBuilder* possui os métodos *execute*() e *measure*().

```
14
15  /**
16   * Class that implement the MeasureBuilder. In this class, the granularity of the measure received by parameter
17   * is verified, in order to redirecting the evaluation for the proper PackageBuilder. Then, the results for each
18   * in the plugin is showed in the measurement view.
19   *
20   * @author Mariana Azevedo
21   * @since 14/11/2014
22   */
23  public class MeasureBuilder {
24
25      private PackageBuilder builder;
26      private ItemMeasured root;
27      private List<Measure> measures;
28
29      public MeasureBuilder(){
30          measures = new ArrayList<Measure>();
31      }
32
33      public void addMeasure(Measure measure){
34          this.measures.add(measure);
35      }
36
37      /**
38       * Method that receives an IProject and 'create' a new project derived. This derived
39       * project is storaged in the AST.
40       * @param project
41       * @return ItemMeasured
42       * @throws CoreException
43       */
44      public ItemMeasured execute(IProject project) throws CoreException {
45          root = null;
46          ASTSession.getInstance().reset();
47          if (project != null && project.isNatureEnabled("org.eclipse.jdt.core.javanature")) {
48              IJavaProject javaProject = JavaCore.create(project);
49              root = measure(javaProject);
50          }
51          return root;
52      }
53
54      /**
55       * Method that invokes all the calculation in projects.
56       * @param javaProject
57       * @return ItemMeasured
58       */
59      private ItemMeasured measure(IJavaProject javaProject) {
60          root = new ItemMeasured(javaProject.getElementName(), null);
61          builder = new PackageBuilder(javaProject);
62          for (Measure m : measures) {
```

Figura 9 *Snapshot* do código da classe *Measurebuilder*
Fonte: (*O3SMeasures*)

No método *execute*(), ASTSession é "resetada" e um projeto Java é recebido como parâmetro para iniciar a análise. O médoto *execute*() invoca o método *measure*(), cuja responsabilidade é iniciar o cálculo das medidas, iterando em cada pacote do projeto (há uma instância da classe *PackageBuilder*, anteriormente explicada, que realiza a avaliação e a apresentação dos resultados da medição, considerando os pacotes do sistema analisado).

6.4 Descrição das medidas

Nesta subseção, é fornecida uma breve descrição de cada medida implementada no *plug-in O3SMeasures*:

a) ***Lines of Code* (LOC) - Linhas de Código:** calcula a quantidade total de linhas de código de um projeto. Não são contabilizados espaços em branco e comentários;
b) ***Number of Classes* (NC) - Número de Classes:** calcula a quantidade total de classes em um projeto (conta também as classes internas);
c) ***Number of Attributes* (NOA) - Número de Atributos:** calcula a quantidade total de atributos em um projeto;
d) ***Number of Methods* (NOM) - Número de Métodos:** calcula a quantidade total de métodos em um projeto;
e) ***Response for a Class* (RFC) - Resposta para uma Classe:** calcula a quantidade de métodos e de construtores distintos invocados por uma classe;
f) ***Coupling Between Objects* (CBO) - Acoplamento entre Objetos:** calcula a quantidade de outros tipos de referência de classe ou de interface, que ocorrem por meio de chamadas de método, os

parâmetros de métodos, os tipos de retorno, as exceções lançadas e os campos acessados em uma dada classe analisada;

g) ***Fan-out* (FOUT):** calcula a quantidade de classes referenciadas por uma classe em um pacote;

h) ***Weight Methods per Class* (WMC) - Peso de Métodos por Classe:** soma a complexidade ciclomática dos métodos em uma classe;

i) ***Cyclomatic Complexity* (CC) - Complexidade Ciclomática:** conta a quantidade de fluxos em um bloco de código. Inclui verificação de laços de repetição, os condicionais (*if*, *else*), os operadores ternários e os operadores lógicos em expressões. Cada vez que essas estruturas são detectadas, a medida é incrementada por um;

j) ***Depth of Inheritance Tree* (DIT) - Profundidade da Árvore de Herança:** calcula a posição/distância de uma classe na árvore de herança ou a hierarquia de um projeto;

k) ***Number of Children* (NOC) - Número de Filhos:** calcula a quantidade total de subclasses diretas de uma classe. Se uma classe implementa uma interface é contada como um filho direto dessa interface;

l) ***Lack of Cohesion Between Methods* (LCOM) - Ausência de Coesão entre métodos:** é calculado utilizando o método de Chidamber e Kemerer. Para cada par de métodos na classe há duas variáveis de decisão a serem incrementadas: P (métodos que acessam conjuntos distintos de variáveis) e Q (métodos que acessam pelo menos uma mesma variável). Se acessarem conjuntos disjuntos de variáveis de instância, aumentar a variável P em um; por outro lado, se compartilharem pelo menos um acesso variável, aumentar Q em um;

m) ***Lack of Cohesion Between Methods 2*** **(LCOM2) - Ausência de Coesão entre métodos 2:** calculado utilizando o método de Henderson-Sellers. Se M(A) é a quantidade de métodos que acessa a um atributo A, é calculado a média de M(A) para os atributos, subtraindo a quantidade de métodos de M e dividido o resultado por (1-M). Um valor baixo indica um classe coesa. Um valor próximo de 1 indica uma classe pouco coesa, sugerindo melhorias podem ser feitas, por exemplo, segmentando a classe com baixa coesão em duas ou mais classes;

n) ***Lack of Cohesion Between Methods 4*** **(LCOM4) - Ausência de Coesão entre métodos 4:** calcula a variação da medida LCOM de CK, LCOM4, proposta por Hitz e Montazeri. Mede a quantidade de "componentes ligados" em uma classe. Um componente ligado é um conjunto de métodos relacionados (e variáveis de nível de classe). Deve haver apenas um componente ligado em cada classe. Se houver dois ou mais componentes, a classe deve ser dividida em outras classes menores;

o) ***Loose Class Cohesion (LCC)*** **- Classe de autonomia fraca:** calcula a quantidade de métodos ligados de forma direta ou indireta em uma classe. É a quantidade de ligações diretas (quantidade de métodos conectados entre si), somado à quantidade de conexões indiretas (quantidade de chamadas de um método dentro de outro método) e divida pela quantidade máxima de possíveis conexões;

p) ***Tight Class Cohesion (TCC)*** **- Classe de autonomia forte:** calcula densidade das ligações. É a quantidade de ligações diretas (quantidade de métodos conectados entre si) dividida pela quantidade máxima de possíveis conexões.

6.5 Funcionamento do *plug-in*

O *plug-in O3SMeasures* pode ser acessado via *menu pop-up* mostrado após selecionar um sistema de software Java no Eclipse IDE, com o botão direito do *mouse* (Figura 10). Após a seleção da opção *Measure*, os valores para as medidas são calculados, cujo progresso pode ser acompanhado por meio de uma caixa de diálogo (Figura 11). O resultado da medição é apresentado na *view O3SMeasures SpreadSheet* (Figura 12) e na *view O3SMeasures PieChart* (Figura 13).

Na *view O3SMeasures SpreadSheet*, são apresentados os resultados da medição em uma *TreeView* para o usuário visualizar os valores totais e a média e a descrição de cada uma das medidas. Por exemplo, para o sistema de software *toy-project*, o valor da medida *Number of Methods* é 17 (Figura 14). A visualização do valor das medidas pode ser expandida selecionando a seta em cada um dos itens. Por exemplo, ao selecionar a medida *Number of Methods*, pode-se visualizar a quantidade de métodos no pacote toyproject.dto (15 pacotes) e a quantidade de métodos na classe Animal.java (nove métodos) (Figura 15). Essas *views* podem ser habilitadas no *menu Window → Show View*, na barra de ferramentas do Eclipse (Figura 16).

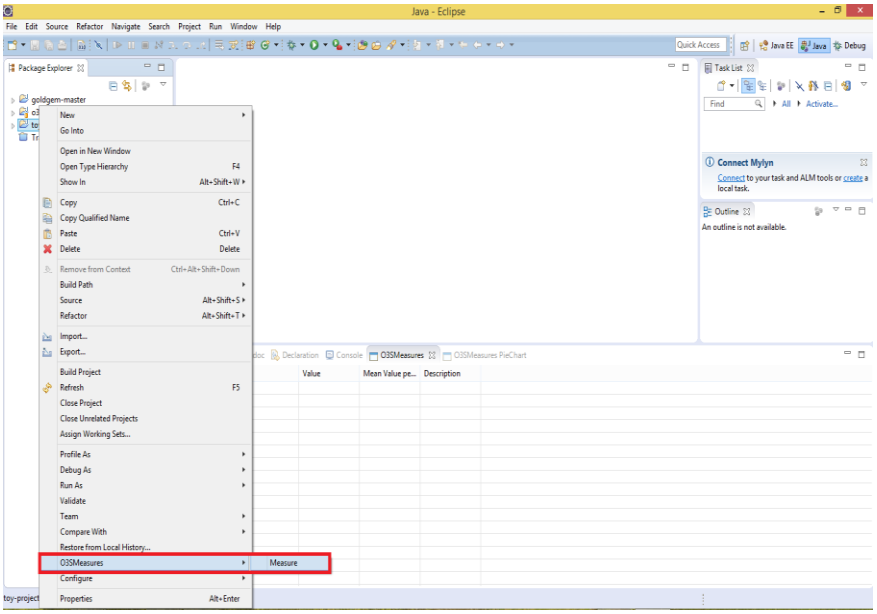


Figura 10 *Menu* de acesso ao *O3SMeasures*

Fonte: (*O3SMeasures*).

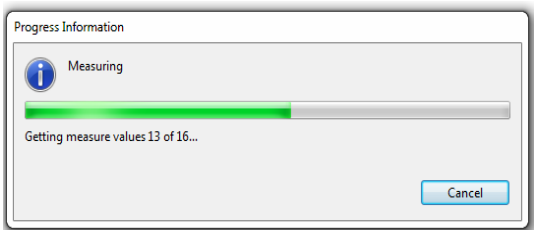
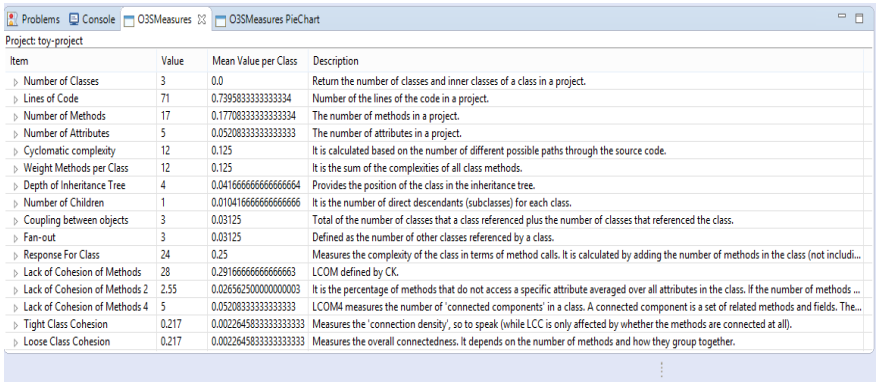


Figura 11 Caixa de diálogo mostrando o progresso da medição

Fonte: (*O3SMeasures*).



Problems | Console | O3SMeasures | O3SMeasures PieChart

Project: toy-project

Item	Value	Mean Value per Class	Description
Number of Classes	3	0.0	Return the number of classes and inner classes of a class in a project.
Lines of Code	71	0.7395833333333334	Number of the lines of the code in a project.
Number of Methods	17	0.17708333333333334	The number of methods in a project.
Number of Attributes	5	0.05208333333333333	The number of attributes in a project.
Cyclomatic complexity	12	0.125	It is calculated based on the number of different possible paths through the source code.
Weight Methods per Class	12	0.125	It is the sum of the complexities of all class methods.
Depth of Inheritance Tree	4	0.041666666666666664	Provides the position of the class in the inheritance tree.
Number of Children	1	0.010416666666666666	It is the number of direct descendants (subclasses) for each class.
Coupling between objects	3	0.03125	Total of the number of classes that a class referenced plus the number of classes that referenced the class.
Fan-out	3	0.03125	Defined as the number of other classes referenced by a class.
Response For Class	24	0.25	Measures the complexity of the class in terms of method calls. It is calculated by adding the number of methods in the class (not includi...
Lack of Cohesion of Methods	28	0.2916666666666663	LCOM defined by CK.
Lack of Cohesion of Methods 2	2.55	0.026562500000000003	It is the percentage of methods that do not access a specific attribute averaged over all attributes in the class. If the number of methods ...
Lack of Cohesion of Methods 4	5	0.05208333333333333	LCOM4 measures the number of 'connected components' in a class. A connected component is a set of related methods and fields. The...
Tight Class Cohesion	0.217	0.0022645833333333333	Measures the 'connection density', so to speak (while LCC is only affected by whether the methods are connected at all).
Loose Class Cohesion	0.217	0.0022645833333333333	Measures the overall connectedness. It depends on the number of methods and how they group together.

Figura 12 *View O3SMeasures SpreadSheet*

Fonte: (*O3SMeasures*).

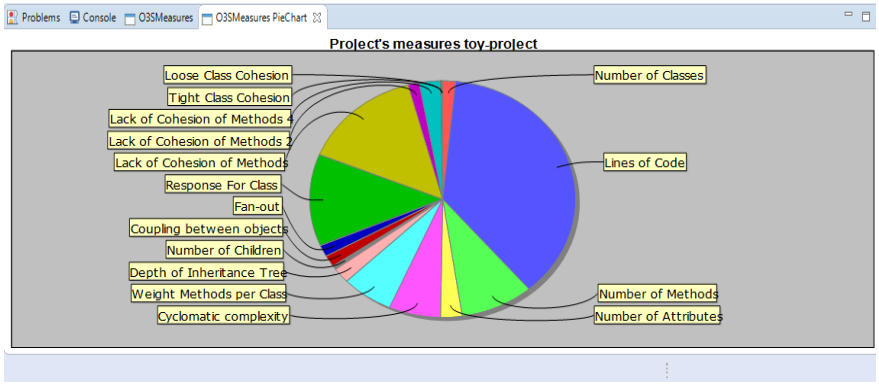
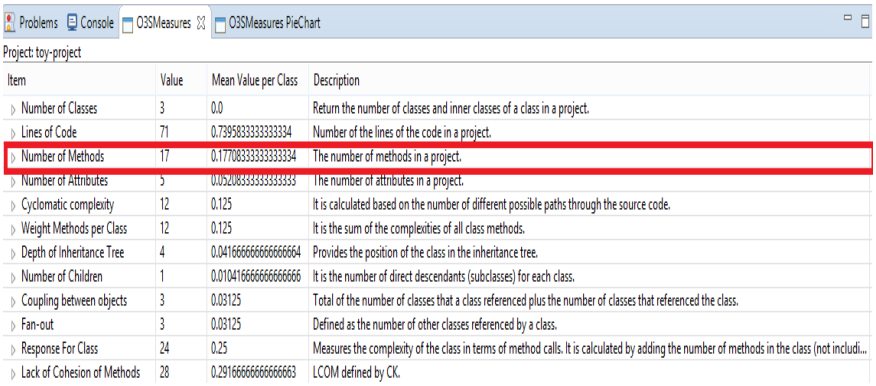


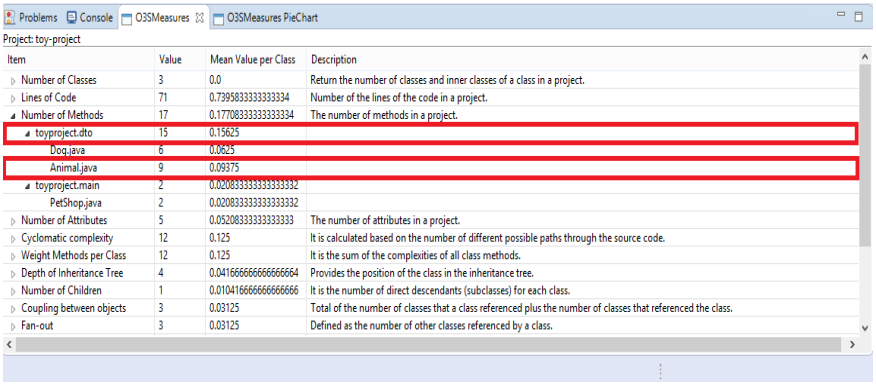
Figura 13 *View O3SMeasures PieChart*

Fonte: (*O3SMeasures*).



Item	Value	Mean Value per Class	Description
Number of Classes	3	0.0	Return the number of classes and inner classes of a class in a project.
Lines of Code	71	0.7395833333333334	Number of the lines of the code in a project.
Number of Methods	17	0.17708333333333334	The number of methods in a project.
Number of Attributes	5	0.05208333333333333	The number of attributes in a project.
Cyclomatic complexity	12	0.125	It is calculated based on the number of different possible paths through the source code.
Weight Methods per Class	12	0.125	It is the sum of the complexities of all class methods.
Depth of Inheritance Tree	4	0.041666666666666664	Provides the position of the class in the inheritance tree.
Number of Children	1	0.010416666666666666	It is the number of direct descendants (subclasses) for each class.
Coupling between objects	3	0.03125	Total of the number of classes that a class referenced plus the number of classes that referenced the class.
Fan-out	3	0.03125	Defined as the number of other classes referenced by a class.
Response For Class	24	0.25	Measures the complexity of the class in terms of method calls. It is calculated by adding the number of methods in the class (not includi...
Lack of Cohesion of Methods	28	0.2916666666666663	LCOM defined by CK.

Figura 14 Resultado da medida *Number of methods* do *toy-project*
Fonte: (*O3SMeasures*).



Item	Value	Mean Value per Class	Description
Number of Classes	3	0.0	Return the number of classes and inner classes of a class in a project.
Lines of Code	71	0.7395833333333334	Number of the lines of the code in a project.
Number of Methods	17	0.17708333333333334	The number of methods in a project.
toyproject.dto	15	0.15625	
Dog.java	6	0.0625	
Animal.java	9	0.09375	
toyproject.main	2	0.020833333333333332	
PetShop.java	2	0.020833333333333332	
Number of Attributes	5	0.05208333333333333	The number of attributes in a project.
Cyclomatic complexity	12	0.125	It is calculated based on the number of different possible paths through the source code.
Weight Methods per Class	12	0.125	It is the sum of the complexities of all class methods.
Depth of Inheritance Tree	4	0.041666666666666664	Provides the position of the class in the inheritance tree.
Number of Children	1	0.010416666666666666	It is the number of direct descendants (subclasses) for each class.
Coupling between objects	3	0.03125	Total of the number of classes that a class referenced plus the number of classes that referenced the class.
Fan-out	3	0.03125	Defined as the number of other classes referenced by a class.

Figura 15 Resultado da medida *Number of methods* por pacote e por classe
Fonte: (*O3SMeasures*).

O usuário pode salvar o resultado da medição em um arquivo com a extensão .csv (itens separados por vírgula) ao selecionar com o botão direito do *mouse* a opção *Export to CSV file* (Figura 17). Esse arquivo é criado no diretório padrão do sistema operacional que o usuário utiliza.

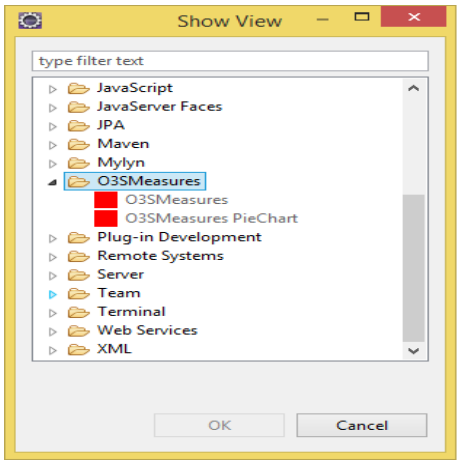


Figura 16 *Menu show view* do Eclipse
Fonte: (*O3SMeasures*).

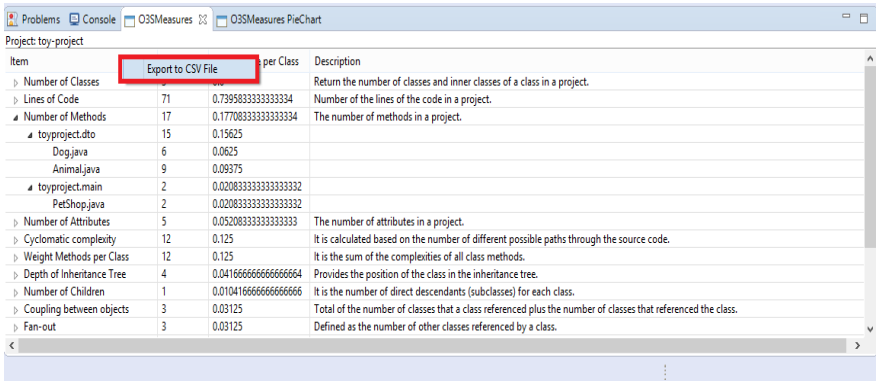


Figura 17 Opção export to *CSV* file
Fonte: (*O3SMeasures*).

6.6 Considerações finais

Medidas têm sido propostas para avaliar a qualidade interna de sistemas de software OO. Várias ferramentas foram desenvolvidas para medir as propriedades de software, tais como, a quantidade de linhas, o acoplamento e a complexidade. No entanto, não foram encontradas ferramentas para medir

software desenvolvido em Java, que possuam em seu catálogo de medidas, por exemplo, as medidas LCOM2, LCOM4 e LCC, consideradas revelantes para a avaliação da qualidade interna de software OO, conforme os resultados obtidos na RSL executada neste trabalho.

Nesta seção foi apresentado um *plug-in* para o Eclipse IDE denominado *O3SMeasures*. Esse *plug-in* implementa as medidas elencadas na RSL (Seção 5). Além disso, foram apresentados exemplos de utilização do *O3SMeasures* e resultados das medições obtidos. A opção de desenvolver o *plug-in* em inglês foi uma decisão de projeto. A parametrização de classes, de funções, de variáveis e de outros elementos do apoio computacional em inglês, tem como benefícios:

a) O código poderá ser mantido por alguém que não compreende o idioma Português;
b) As APIs quase sempre estão em inglês, pois há interesse dos autores em compartilhar o apoio computacional desenvolvido com a comunidade de software internacional.

7 ANÁLISE DESCRITIVA - CARACTERIZAÇÃO DE DOMÍNIOS DE SISTEMAS DE SOFTWARE

7.1 Considerações iniciais

A estatística descritiva é um ramo da estatística cujo objetivo é aplicar técnicas para organizar, descrever e sumarizar um conjunto de dados (HAIR et al., 2009). Dentro da estatística descritiva, existem algumas medidas normalmente utilizadas para descrever um conjunto de dados, por exemplo, as **medidas de tendência central** e as **medidas de variabilidade ou dispersão**. As medidas de tendência central incluem a média, a mediana e a moda. As medidas de variabilidade incluem o desvio padrão, a variância, o valor máximo e o valor mínimo. Nesta seção, a descrição e a sumarização dos dados utilizados para a obtenção do modelo multivariado são apresentadas considerando os domínios dos sistemas de software selecionados.

Esta seção está organizada da seguinte forma. A caracterização da amostra é feita na subseção 7.2. A caracterização dos domínios estudados é feita na subseção 7.3. A discussão dos resultados obtidos é apresentada na subseção 7.4.

7.2 Caracterização da amostra

O *dataset* do estudo foi constituído por sistemas de software *open-source* desenvolvidos em Java, selecionados a partir de Github e Sourceforge, dois repositórios mais populares da *web*. A coleta de dados foi realizada no período de 01/10/13 a 31/12/14. O Sourceforge é um repositório para o desenvolvimento e distribuição de software de código aberto, que possui mais de 430 mil projetos e hospeda mais de 3.7 milhões de usuários registrados

(Sourceforge, 2015). O Github é um serviço de hospedagem compartilhado para sistemas de software de código aberto que utilizam o controle de versionamento Git[11]. Oferece dois planos pagos (*Personal* e *Organizational*) para repositórios privados e contas gratuitas. Em 2014, o Github relatou ter mais de 3.4 milhões de usuários e com 16.7 milhões de projetos (GITHUB, 2015). Os critérios inclusão de sistemas de software no *dataset* foram:

a) Sistemas de software escritos em Java;
b) Sistemas de software compiláveis;
c) Sistemas de software cuja última versão disponível foi desenvolvida entre 2013-2014.

Foi coletado o total de 1.031 sistemas de software e categorizados em: i) *Audio & Video* (105 sistemas); ii) *Business & Enterprise* (124 sistemas); iii) *Communications* (112 sistemas); iv) *Development* (132 sistemas); v) *Home & Education* (80 sistemas); vi) *Games* (112 sistemas); vii) *Graphics* (89 sistemas); viii) *Science & Engineering* (93 sistemas); ix) *Security & Utilities* (83 sistemas); e x) *System Administration* (101 sistemas). Para cada uma das 10 categorias ou domínios de software propostos pelo Sourceforge, foram selecionados aleatoriamente, sistemas de software para serem medidos, seguindo o fator de proporcionalidade populacional

$$n_i = (N_i/N) \times n$$

Sendo n_i o tamanho da amostra no estrato i, N_i o tamanho do estrato populacional i, N o tamanho total da população em estudo e n o tamanho total da amostra coletada para cada domínio.

[11] http://git-scm.com/

Dessa forma, a amostra utilizada no estudo é composta por 500 sistemas de software em que 51 sistemas de *Audio & Video*, 60 sistemas de *Business & Enterprise*, 54 sistemas de *Communications*, 64 sistemas de *Development*, 39 sistemas de *Home & Education*, 54 sistemas de *Games*, 43 sistemas de *Graphics*, 45 sistemas de *Science & Engineering*, 40 sistemas de *Security & Utilities* e 49 sistemas de *System Administration*. Essa amostra possui em torno de 229.170 classes e totalizam mais de 20.838.192 de linhas de código. Os sistemas de software do Github foram organizados nos domínios com base na descrição em seu repositório, uma vez que o repositório não categoriza os sistemas por domínios, mas por linguagem de programação. Os dez domínios propostos pelo Sourceforge são:

a) ***Audio & Video* (AV)**. Nessa categoria, compreendem as soluções de áudio como análise de som/áudio, de captura/gravação, de conversão de áudio, os mixadores, os editores, os *players*, os sistemas de síntese sonora, software de captura de discurso (*speech*, MIDI e composição) e vídeo, por exemplo, software de captura de vídeo, de conversão de formatos, de efeitos-especiais, de pacotes de *codecs*, de processamento de vídeo em tempo real, de edição não linear e de visualização de vídeos;
b) ***Business & Enterprise* (BE)**. Nessa categoria, compreendem as soluções voltadas para negócios e gestão de empresas em geral, considerando sistemas de aplicações financeiras ou de gestão financeira (de investimentos, de orçamento, de contabilidade, de gestão financeira pessoal), as suítes de aplicações de escritório (*Office suites*), os geradores de relatórios, os sistemas de modelagem de negócios, de gestão do conhecimento, os sistemas *e-commerce*, de gestão de projetos, de aplicações com listas de tarefas a fazer (*To-do*

Lists) e de gestão empresarial nos aspectos de qualidade, de escopo e de prazos;

c) ***Communication* (C)**. Nessa categoria, compreendem as soluções de comunicação, por exemplo, sistemas de telefonia, de transferência de dados (*streaming*), de sincronização de dados, de leitores de *feed* (RSS), as soluções de *e-mail*, de conferência, de rádio e FIDO;

d) ***Development* (D)**. Nessa categoria, compreendem as soluções para o desenvolvimento de software em geral. São incluídos as IDEs, os *debuggers*, os compiladores, os interpretadores, os sistemas de geração de código automático, de controle de versão, de modelagem da arquitetura de sistemas (OO, modelagem de bancos de dados), de geração de documentação, de garantia da qualidade, de teste, de criação de interfaces para o usuário e de garantia da usabilidade, os *frameworks*, as máquinas virtuais (*Virtual Machines* - VMs), os sistemas e os *plug-ins* de análise e de revisão de código, as bibliotecas, os servidores, os *templates*, os editores binários, as ferramentas CASE, os editores de texto e os sistemas de banco de dados;

e) ***Games* (G)**. Nessa categoria, compreendem os sistemas de software classificados como jogos (2D e 3D), podendo ser de estratégia em tempo real, os jogos em primeira pessoa, de multiusuários (*multiplayer*s), de simulação, os ogos de tabuleiro, os quebra-cabeças, os jogos de cartas, os arcades, os jogos esportivos, os jogos baseados em consoles específicos e os *frameworks* de desenvolvimento de jogos;

f) ***Graphics* (GPH)**. Nessa categoria, compreendem soluções de captura de imagens, de conversão gráfica, os editores de imagem, de modelagem 3D, de renderização 3D, os sistemas de apresentação, os

visualizadores, as galerias de imagens, sistemas de animação, de reconhecimento de grafia, de gerações procedurais e os fractais;

g) ***Home & Education*** **(HE)**. Nessa categoria, compreendem soluções educativas como sistemas voltados para a educação social, religiosa, filosófica e de grande utilidade para a sociedade, como geradores de arquivos nos formatos pdf, TeX e os sistemas de impressão;

h) ***Science & Engineering*** **(SE)**. Nessa categoria, compreendem soluções de engenharia e científicas. São incluídos sistemas de matemática, de estatística, de física e de química, de inteligência artificial, de astronomia, de bioinformática, de ciências médicas e biológicas, de ciência molecular, de linguística, de robótica, de engenharia civil, de teste e de medição;

i) ***Security & Utilities*** **(SU)**. Nessa categoria, compreendem soluções de arquivamento, de segurança, de protocolos de transmissão de arquivos, de análises e rotação de *log*, os terminais e os gerenciadores de arquivos;

j) ***System Administration*** **(SA)**. Nessa categoria, compreendem soluções para administração de um sistema de informação. São incluídos sistemas de *boot*, de *clustering* de máquinas, de sistemas distribuídos, de sistemas em *shell*, de sistemas operacionais, de sistemas de *benchmark*, de busca, de armazenamento, de *logging*, os instaladores, os sistemas embarcados, sistemas de automação e sistemas para hardware em geral.

Para descrever e caracterizar a amostra, foram utilizadas medidas de tendência central e medidas de variabilidade ou dispersão. O valor médio das medidas utilizadas é apresentado na Tabela 9, na Tabela 10 e na Tabela 11 em

que o maior valor está grafado em negrito e o menor valor está grafado em itálico e sublinhado.

Tabela 9 Valor médio das medidas selecionadas - conjunto I

Domínios	NC	LOC	NOM	NOA	CC	WMC
AV	345,784	116,85902	8,07358	**6,62139**	52,39213	**29,56788**
BE	**986,230**	102,33277	7,83013	4,56531	41,05184	17,53737
C	264,130	92,32413	7,07028	4,98575	37,67419	16,31426
D	605,422	*84,90805*	7,79116	*3,48535*	35,66790	16,22414
G	357,852	103,99421	7,75588	5,08267	44,44322	20,04409
GPH	344,953	115,99873	**8,90568**	5,28827	52,94581	25,76690
HE	502,923	111,75000	8,15118	5,57862	49,15260	24,25763
SE	457,822	100,88259	8,14621	4,43973	41,11949	20,23974
SU	343,275	**125,30261**	8,00740	5,83085	**55,04312**	22,44283
SA	*248,122*	85,41202	*6,86769*	4,06657	*34,03356*	*12,22513*

Tabela 10 Valor médio das medidas selecionadas - conjunto II

Domínios	DIT	NOC	CBO	FOUT	RFC
AV	1,42742	0,60564	1,47548	0,84717	16,29805
BE	1,77405	**1,92222**	1,44903	**0,88939**	14,15628
C	1,42807	0,45964	1,40515	0,85794	*13,14418*
D	*1,40380*	1,02502	*1,28241*	0,82370	16,60411
G	**1,87571**	0,66104	1,54819	0,85227	14,82731
GPH	1,71289	1,01652	1,52418	0,82303	**18,68977**
HE	1,72305	0,92213	**1,59178**	0,87758	15,14647
SE	1,67802	1,07690	1,50879	0,83295	17,51609
SU	1,67015	*0,45371*	1,50569	*0,81556*	16,60823
AS	1,47242	0,62166	1,38538	0,87989	13,97390

Esses valores indicam que sistemas do domínio *Business & Enterprise* tendem a ter uma maior quantidade de classes, seguidos dos domínios de *Development*, *Home & Education*, *Science & Engineering* e *Games*. Além disso, os sistemas do domínio *Business & Enterprise* tendem a ter uma maior quantidade de subclasses imediatas na árvore de herança, um maior acoplamento

entre pacotes (valor maior para **FOUT**) e tendem a ser menos coesos (valores maiores para as medidas **LCOM**, **LCOM2** e **LCOM4**; valores menores para **TCC** e **LCC**).

Tabela 11 Valor médio das medidas selecionadas - conjunto III

Domínios	LCOM	LCOM2	LCOM4	TCC	LCC
AV	389,06167	0,80950	230,26948	0,13535	0,13536
BE	**933,07727**	**0,84839**	**504,40781**	0,10410	0,10410
C	271,69841	0,81287	151,79465	0,10074	*0,10074*
D	455,07199	0,81123	218,36583	**0,9850**	**0,9850**
G	344,953	0,81629	238,68298	0,15912	0,15912
GPH	400,79145	*0,78743*	228,55850	0,10950	0,10950
HE	523,16457	0,83521	308,89431	0,11535	0,11535
SE	455,85430	0,81507	252,41815	0,10381	0,10381
SU	375,16197	0,80151	223,92838	*0,09879*	0,13104
AS	*245,31093*	0,80385	*128,44805*	0,115261	0,115261

Os sistemas nos domínios *Communication, Development, Business & Enterprise* e *System Administration* tendem a ser menos complexos (valores menores para as medidas **WMC** e **CC**, respectivamente). Em relação à propriedade herança, os sistemas do domínio *Games* tendem a ter a maior hierarquia entre classes, seguido dos domínios *Business & Enterprise, Home & Education, Graphics e Science & Engineering*; em contrapartida, sistemas no domínio *Development* tendem a ter menor hierarquia entre classes, assim como os domínios *Audio & Video, Communication*, *System Administration e Security & Utilities*.

Os sistemas de software dos domínios *Home & Education* e *Graphics* tendem a ser mais acoplados (valores maiores para as medidas **CBO** e **RFC**, respectivamente). Os sistemas do domínio *Security & Utilities* também apresentam o mesmo comportamento que os domínios *Home & Education* e *Graphics* para essas medidas. Quanto à coesão, os domínios *Development*,

Games e *Audio & Video* tendem a ser mais coesos (valores maiores para as medidas **TCC** e **LCC**, respectivamente). Esse resultado indica que as classes dos sistemas desses domínios possuem responsabilidades bem definidas (representam apenas um conceito e funções específicas para as entidades). Além disso, para que um código seja reutilizável, é necessário que suas funções sejam bem definidas para a alteração no código em uma área não afetar o código em outra área.

Com o resultado preliminar da análise descritiva, percebe-se que o comportamento das medidas varia de domínio para domínio. Assim como nessa primeira análise, alguns trabalhos propõem medir propriedades de qualidade de software, utilizando famílias de distribuições que seguem uma Lei de Potência (SIMON, 1955; BAXTER et al., 2006; FERREIRA et al., 2012). Quando a média aritmética das medidas segue uma Lei de Potência, por exemplo, a distribuição e a frequência do valor das medidas possuem um formato de cauda longa (SIMON, 1955). Essa característica de distribuição descrita, com a presença de uma curva no formato de cauda longa, pode ser observada na Figura 18.

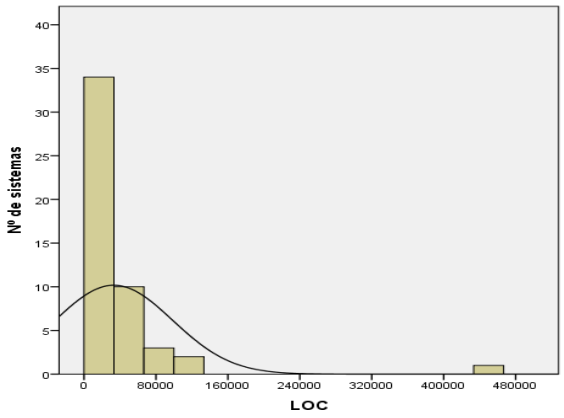


Figura 18 Distribuição de linhas de código (Medida LOC) para o domínio *audio & video*

Além disso, observa-se que, para a medida **LOC**, a maioria dos sistemas do domínio *Audio & Video* possui menos do que 60.000 linhas de código; por outro lado, há poucos sistemas com mais de 100.000 linhas de código. Essa característica da cauda longa pode indicar que os dados coletados não são normalmente distribuídos. Assim, para verificar tal comportamento, foi executado um teste de *Kolmogorov-Smirnov*, cujos valores são apresentados na Tabela 12.

Tabela 12 Teste de normalidade da amostra

Medidas	**Kolmogorov-Smirnov[a]**		
	Estatísticas	***df***	**Valor de *p***
NC	0,289	500	0,000
LOC	0,278	500	0,000
NOM	0,110	500	0,000
NOA	0,150	500	0,000
CC	0,185	500	0,000
WMC	0,206	500	0,000
DIT	0,074	500	0,000
NOC	0,377	500	0,000
CBO	0,176	500	0,000
FOUT	0,159	500	0,000
RFC	0,140	500	0,000
LCOM	0,259	500	0,000
LCOM2	0,126	500	0,000
LCOM4	0,251	500	0,000
TCC	0,163	500	0,000
LCC	0,182	500	0,000

[a] Correção da significância de *Lilliefors*

Na Tabela 12, pode-se observar um resumo descritivo das variáveis em estudo. O resultado do teste indica que, com a probabilidade de erro de 1%, a amostra não é normalmente distribuída ($p < 0,001$). Cabe ressaltar que a análise fatorial e o PLS-SEM não exigem uma amostra normalmente distribuída (CHIN, 1998; CHIN; MARCOLIN; NEWSTED, 2003; HAIR et al., 2009; HAIR et al.,

2014). Dessa forma, na próxima subseção, as características de qualidade interna dos domínios são apresentadas.

7.3 Caracterização dos domínios de sistemas de software

A estatística descritiva para os sistemas do domínio *Audio & Video* (AV), incluindo o mínimo, o 1º quartil (25%), a mediana, o 3º quartil (75%), o máximo e o desvio padrão, é apresentada na Tabela 13. Por exemplo, em relação ao tamanho, os sistemas do domínio AV tendem a ter mínimo de quatro e máximo de 4.582 classes. O quartil inferior (1/4 da amostra) está entre quatro a 32 classes. O quartil superior (3/4 da amostra) está entre 467 a 4.582 classes. Esses sistemas possuem de 245 a 440.738 linhas de código. Além disso, a quantidade de métodos varia de 0,001 a 23 métodos por classe e a quantidade de atributos varia de 0,002 a 27 atributos por classe.

Tabela 13 Estatística descritiva para o domínio *Audio & Video*

Medidas	Mínimo	Máximo	25%	Mediana	75%	Desvio Padrão
NC	4	4582	32	131	467	676,171
LOC	245	440738	3600	12970	43575	64652,117
NOM	0,001	23	5,201	7,724	9,527	4,555
NOA	0,002	27	3,799	4,745	8,086	5,129
CC	0,011	195,058	23,026	35,587	70,463	44,788
WMC	0,008	318,500	10,357	16,925	34,851	46,346
DIT	0	3,781	1,063	1,463	1,927	0,866
NOC	0	2,522	0	0,345	1,0	0,688
CBO	0,231	7,136	1,107	1,338	1,727	0,941
FOUT	0,019	2,591	0,794	0,897	0,938	0,335
RFC	2,462	60,222	8,947	13,726	19,752	11,508
LCOM	2,538	4150,871	42,489	194,328	445,287	645,290
LCOM2	0,226	0,998	0,775	0,879	0,922	0,181
LCOM4	2,385	2176,069	33,878	136,046	239,286	353,881
TCC	0,374	0,903	0,066	0,105	0,185	0,091
LCC	0,374	0,903	0,066	0,105	0,185	0,091

Ainda na Tabela 13, quanto às medidas de complexidade, a medida CC varia de 0,011 a 195,058 por classe. A medida WMC varia de 0,008 a 318,500 por classe. Quanto às medidas de herança, a medida DIT varia de 0,0 a 3,781 por classe (em que a maior hierarquia entre classes é de aproximadamente 4,0). A medida NOC varia de 0,0 a 2,522 por classe, ou seja, a menor quantidade de subtipos em uma classe é 0,0, e a maior quantidade de subtipos em uma classe é de aproximadamente 3,0. Quanto às medidas de acoplamento, a medida CBO varia de 0,231 a 7,136 por classe; a medida FOUT varia de 0,019 a 2,591 por classe; e a medida RFC varia de 2,462 a 60,222 por classe. Quanto às medidas de coesão, a medida LCOM varia de 2,538 a 4150,871 por classe; LCOM2 varia de 0,226 a 0,998 por classe; LCOM4 varia de 2,385 a 2176,069 por classe; e as medidas TCC e LCC variam de 0,374 a 0,903 por classe.

Para os sistemas do domínio *Business & Enterprise* (BE), a estatística descritiva é apresentada na Tabela 14. Em relação ao tamanho, os sistemas do domínio BE tendem a ter mínimo de 5,0 e máximo de 5.290 classes. O quartil inferior (1/4 da amostra) está entre 5,0 a 48 classes. O quartil superior (3/4 da amostra) está entre 1.420,5 a 5.290 classes. Esses sistemas possuem de 266 a 509.058 linhas de código. Além disso, a quantidade de métodos varia de 3,125 a 17,79 métodos por classe e a quantidade de atributos varia de 1,0 a 14,8 atributos por classe. Quanto às medidas de complexidade, a medida CC varia de 2,596 a 212,648 por classe. A medida WMC varia de 1,426 a 58,4 por classe.

Tabela 14 Estatística descritiva para o domínio *Business & Enterprise*

Medidas	Mínimo	Máximo	25%	Mediana	75%	Desvio Padrão
NC	5,0	5.290	48	388	1.420,5	1.280,507
LOC	266	509.058	4.383,50	45.344	143.818,5	117.837,843
NOM	3,125	17,79	6,004	7,11429	9,06706	2,678953
NOA	1,0	14,8	2,77283	3,85797	5,48316	2,710139
CC	2,596	212,648	22,20091	30,73298	42,265	37,085552
WMC	1,426	58,4	10,48998	14,24401	21,74	11,943487
DIT	0	3,436	1,32213	1,70330	2,26923	0,615814
NOC	0	61	0,44890	0,68750	1,19706	7,732292
CBO	0,931	2,8	1,14416	1,31746	1,66264	0,424636
FOUT	0,531	1,0	0,84172	0,92624	0,96790	0,109051
RFC	5,625	30,198	10,02317	13,96606	17,137	5,384422
LCOM	3,0	4.975,28	58,42554	507,7668	1.454,882	1.159,698
LCOM2	0,465	0,991	0,77679	0,89760	0,94383	0,126009
LCOM4	4,5	3.338,57	37,11818	220,2650	699,4	656,5488
TCC	0	0,3	0,07221	0,09691	0,13410	0,060919
LCC	0	0,3	0,07221	0,09691	0,13410	0,060919

Em relação às medidas de herança, a medida DIT varia de 0,0 a 3,436 por classe (em que a maior hierarquia entre classes é de aproximadamente 3,0). A medida NOC varia de 0,0 a 61 por classe, ou seja, a menor quantidade de subtipos em uma classe é 0,0 e a maior quantidade de subtipos em uma classe é de 61. Quanto às medidas de acoplamento, a medida CBO varia de 0,931 a 2,8 por classe; a medida FOUT varia 0,531 a 1,0 por classe; e a medida RFC varia de 5,625 a 30,198 por classe. Quanto às medidas de coesão, a medida LCOM varia de 3,0 a 4.975,28 por classe; LCOM2 varia de 0,226 a 0,991 por classe; LCOM4 varia de 4,5 a 3.338,57 por classe; e as medidas TCC e LCC variam de 0,0 a 0,3 por classe.

A estatística descritiva dos sistemas do domínio *Communication* (C) é apresentada na Tabela 15. Para as medidas de tamanho, os sistemas do domínio C tendem a ter mínimo de 3,0 e máximo de 3.449 classes. O quartil inferior (1/4 da amostra) está entre 3,0 a 24 classes. O quartil superior (3/4 da amostra) está

entre 284,25 a 3.449 classes. Esses sistemas possuem de 264 a 298.778 linhas de código. Além disso, a quantidade de métodos varia de 1,615 a 17 métodos por classe e a quantidade de atributos varia de 0,444 a 14,154 atributos por classe.

Tabela 15 Estatística descritiva para o domínio *Communication*

Medidas	Mínimo	Máximo	25%	Mediana	75%	Desvio Padrão
NC	3,0	3.449	24	101	284,25	516,7804
LOC	264	298.778	1.955,75	8331,5	27.592	45.517,5965
NOM	1,615	17	5,20748	6,46519	8,31419	3,014361
NOA	0,444	14,154	3,01238	4,51283	6,05281	2,902903
CC	3,348	139,167	19,11143	30,07261	47,0548	26,791738
WMC	2,391	69,833	9,98978	13,86566	19,3962	11,104862
DIT	0	4,0	1,0	1,45393	1,85593	0,839191
NOC	0	1,949	0,02593	0,24636	0,78823	0,512409
CBO	0,5	3,8	1,06435	1,24882	1,46626	0,617872
FOUT	0,538	1,0	0,76865	0,87777	0,95956	0,122693
RFC	2,913	30,17	8,86302	12,66652	17,2543	6,313437
LCOM	0,462	3.192,84	46,946	110,70966	273,555	490,569070
LCOM2	0,216	0,997	0,727	0,84840	0,93194	0,162171
LCOM4	0,826	1592,43	26,865	62,90736	166,432	251,430968
TCC	0,001	0,488	0,5613	0,07946	0,13317	0,078217
LCC	0,001	0,488	0,5613	0,07946	0,13317	0,078217

Quanto às medidas de complexidade, a medida CC varia de 3,348 a 139,167 por classe. A medida WMC varia de 2,391 a 69,833 por classe. Em relação às medidas de herança, a medida DIT varia de 0,0 a 4,0 por classe (em que a maior hierarquia entre classes é de aproximadamente 4,0). A medida NOC varia 0,0 a 1,949 por classe, ou seja, a menor quantidade de subtipos em uma classe é 0,0 e a maior quantidade de subtipos em uma classe é de aproximadamente 2,0. Quanto às medidas de acoplamento, a medida CBO varia de 0,5 a 3,8 por classe; a medida FOUT varia de 0,538 a 1 por classe; e, a medida RFC varia de 2,913 a 30,17 por classe. Quanto às medidas de coesão, a medida LCOM varia de 0,462 a 3.192,84 por classe; LCOM2 varia de 0,216 a

0,997 por classe; LCOM4 varia de 0,826 a 1.592,43 por classe; e as medidas TCC e LCC variam de 0,001 a 0,488 por classe.

Para os sistemas do domínio *Development* (D), a estatística descritiva é apresentada na Tabela 16. Os sistemas do domínio D tendem a ter uma quantidade mínima de 4,0 e máximo de 8040 classes. O quartil inferior (1/4 da amostra) está entre 4,0 a 53,5 classes. O quartil superior (3/4 da amostra) está entre 779,75 a 8.040 classes.

Tabela 16 Estatística descritiva para o domínio *Development*

Medidas	Mínimo	Máximo	25%	Mediana	75%	Desvio Padrão
NC	4,0	8.040	53,5	195	779,75	1.145,1226
LOC	229	430.124	4.806,75	19.409	60.642	69.164,3924
NOM	1,958	36,861	5,49301	7,13393	9,074	4,618309
NOA	1,016	9,356	2,39248	2,98182	4,144	1,801174
CC	3,6	162,27	15,582	26,57169	44,898	29,278104
WMC	2,04	91,75	8,2295	10,90225	17,6248	15,140978
DIT	0,006	2,915	1,14437	1,34991	1,73929	0,492749
NOC	0	3,745	0,42513	0,78709	1,35892	0,826376
CBO	0,487	3,806	1,01887	1,17164	1,36012	0,50039
FOUT	0,335	1,067	0,72783	0,85506	0,94851	0,159024
RFC	3,143	71,291	9,52111	13,64007	19,301	11,714990
LCOM	8,4	4.727,91	50,87	219,295	685,631	732,114808
LCOM2	0,064	0,989	0,72269	0,88208	0,93493	0,166964
LCOM4	6,08	2.308,42	30,399	129,397	280,808	341,744595
TCC	0,306	0,896	0,04366	0,18	0,42437	0,172523
LCC	0,306	0,896	0,04366	0,18	0,42437	0,172523

Esses sistemas possuem de 229 a 430.124 linhas de código. Além disso, a quantidade de métodos varia de 1,958 a 36,861 métodos por classe e a quantidade de atributos varia de 1,016 a 9,356 atributos por classe. Quanto às medidas de complexidade, a medida CC varia de 3,6 a 162,27 por classe. A medida WMC varia de 2,04 a 91,75 por classe. Em relação às medidas de herança, a medida DIT varia de 0,006 a 2,915 por classe (em que a maior

hierarquia entre classes é de aproximadamente 3,0). A medida NOC varia 0,0 a 3,745 por classe, ou seja, a menor quantidade de subtipos em uma classe é 0,0 e a maior quantidade de subtipos em uma classe é de aproximadamente 4,0. Quanto às medidas de acoplamento, a medida CBO varia de 0,487 a 3,806 por classe; a medida FOUT varia de 0,335 a 1,067 por classe; e a medida RFC varia de 3,143 a 71,291 por classe. Quanto às medidas de coesão, a medida LCOM varia de 8,04 a 4.727,91 por classe; LCOM2 varia de 0,064 a 0,989 por classe; LCOM4 varia de 6,08 a 2.308,42 por classe; e as medidas TCC e LCC variam de 0,306 a 0,896 por classe.

Para os sistemas do domínio dos Games (G), a estatística descritiva é apresentada na Tabela 17. Em relação ao tamanho, os sistemas do domínio G tendem a ter mínimo de 2,0 e máximo de 3.072 classes. O quartil inferior (1/4 da amostra) está entre 2,0 a 24 classes. O quartil superior (3/4 da amostra) está entre 419,25 a 3.072 classes. Esses sistemas possuem de 128 a 3.644.005 linhas de código. Além disso, a quantidade de métodos varia de 1,0 a 17,338 métodos por classe e a quantidade de atributos varia de 0,556 a 15,175 atributos por classe. Quanto às medidas de complexidade, a medida CC varia de 7,847 a 134,326 por classe. A medida WMC varia de 4,5 a 69 por classe.

Em relação às medidas de herança, a medida DIT varia de 0,0 a 6,0 por classe (em que a maior hierarquia entre classes é 6,0). A medida NOC varia de 0,0 a 3,053 por classe, ou seja, a menor quantidade de subtipos em uma classe é zero e a maior quantidade de subtipos em uma classe é de aproximadamente 3,0. Quanto às medidas de acoplamento, a medida CBO varia de 0,667 a 4,186 por classe; a medida FOUT varia de 0,333 a 1,2 por classe; e a medida RFC varia de 2,0 a 42,7 por classe. Quanto às medidas de coesão, a medida LCOM varia de 0,0 a 3538,692 por classe; LCOM2 varia de 0,0 a 0,997 por classe; LCOM4 varia de 1,0 a 1469,418 por classe; e as medidas TCC e LCC variam de 0,0 a 0,781 por classe.

Tabela 17 Estatística descritiva para o domínio *Games*

Medidas	Mínimo	Máximo	25%	Mediana	75%	Desvio Padrão
NC	2,0	3072	24	99,5	419,25	626,279
LOC	128	3.644.005	2.785,25	10.740,5	42.038,75	66.560,262
NOM	1,0	17,338	6,09099	7,28421	9,80571	3,075626
NOA	0,556	15,175	2,8449	4,37618	6,34362	2,968846
CC	7,847	134,326	23,2273	33,1925	59,74284	30,27724
WMC	4,5	69	8,46469	15,4135	26,31064	15,47989
DIT	0	6,0	1,3871	1,70735	2,13698	0,95993
NOC	0	3,053	0,19167	0,45883	0,8391	0,749291
CBO	0,667	4,186	1,18272	1,36095	1,93611	0,588388
FOUT	0,333	1,2	0,75937	0,86935	0,98077	0,159988
RFC	2,0	42,7	9,95223	14,2484	18,68382	7,31659
LCOM	0	3.538,69	42,4636	148,127	518,0849	707,9727
LCOM2	0	0,997	0,72371	0,84158	0,95762	0,180132
LCOM4	1,0	1469,42	37,1165	98,1046	304,0848	300,5078
TCC	0	0,781	0,05388	0,09704	0,16951	0,179621
LCC	0	0,781	0,05388	0,09704	0,16951	0,179621

A estatística descritiva dos sistemas do domínio *Graphics* (GPH) é apresentada na Tabela 18. Em relação ao tamanho, os sistemas do domínio GPH tendem a ter mínimo de 12 e máximo de 2.689 classes. O quartil inferior (1/4 da amostra) está entre 12 a 46 classes. O quartil superior (3/4 da amostra) está entre 507 a 2.689 classes. Esses sistemas possuem de 1.078 a 300.273 linhas de código. Além disso, a quantidade de métodos varia de 3,776 a 28,515 métodos por classe e a quantidade de atributos varia de 1,687 a 12,606 atributos por classe. Quanto às medidas de complexidade, a medida CC varia de 14,935 a 174,848 por classe. A medida WMC varia de 6,075 a 84,862 por classe.

Em relação às medidas de herança, a medida DIT varia de 0,0 a 3,505 por classe (em que a maior hierarquia entre classes é de aproximadamente 4,0). A medida NOC varia de 0,0 a 13,745 por classe, ou seja, a menor quantidade de subtipos em uma classe é 0,0 e a maior quantidade de subtipos em uma classe é de aproximadamente 14. Quanto às medidas de acoplamento, a medida CBO

varia de 0,75 a 4,036 por classe; a medida FOUT varia de 0,52 a 1,034 por classe; e a medida RFC varia de 5,75 a 93,303 por classe. Quanto às medidas de coesão, a medida LCOM varia de 3,4 a 2008,513 por classe; LCOM2 varia de 0,304 a 0,97 por classe; LCOM4 varia de 5,867 a 1175,175 por classe; e as medidas TCC e LCC variam de 0,0 a 0,525 por classe.

Tabela 18 Estatística descritiva para o domínio *Graphics*

Medidas	Mínimo	Máximo	25%	Mediana	75%	Desvio Padrão
NC	12	2689	46	120	507	491,0088
LOC	1.078	300.273	5.350	12.883	47.195	58.958,195
NOM	3,776	28,515	5,9393	7,1842	10,598	5,0146
NOA	1,687	12,606	3,196	4,516	6,56579	2,901499
CC	14,935	174,848	26,103	38,943	68,714	41,16205
WMC	6,075	84,862	11,933	18,409	30,586	20,57857
DIT	0	3,505	1,3902	1,7328	2,114	0,670107
NOC	0	13,745	0,314	0,514	1,04094	2,089792
CBO	0,75	4,036	1,133	1,31	1,65	0,650212
FOUT	0,52	1,034	0,746	0,831	0,933	0,136782
RFC	5,75	93,303	9,6667	13,44	20,6703	16,09738
LCOM	3,4	2.008,513	76,481	169,354	476,677	531,5923
LCOM2	0,304	0,97	0,7131	0,80123	0,908	0,15227
LCOM4	5,867	1.175,175	61,444	101,925	263,057	286,5626
TCC	0	0,525	0,0554	0,09197	0,13427	0,088995
LCC	0	0,525	0,0554	0,09197	0,13427	0,088995

A estatística descritiva dos sistemas do domínio *Home & Education* (HE) é apresentada na Tabela 19. Em relação ao tamanho, os sistemas do domínio HE tendem a ter mínimo de 30 e máximo de 3.367 classes. O quartil inferior (1/4 da amostra) está entre 30 a 82 classes. O quartil superior (3/4 da amostra) está entre 698 a 3.367 classes. Esses sistemas possuem de 1.791 a 225.849 linhas de código. Além disso, a quantidade de métodos varia de 4,484 a 20,829 métodos por classe e a quantidade de atributos varia de 1,594 a 16,843

atributos por classe. Quanto às medidas de complexidade, a medida CC varia de 14,42 a 241,507 por classe. A medida WMC varia de 3,515 a 83,279 por classe.

Em relação às medidas de herança, a medida DIT varia de 0,0 a 5,947 por classe (em que a maior hierarquia entre classes é de aproximadamente 6,0). A medida NOC varia de 0,0 a 6,067 por classe, ou seja, a menor quantidade de subtipos em uma classe é 0,0 e a maior quantidade de subtipos em uma classe é de aproximadamente 6,0. Quanto às medidas de acoplamento, a medida CBO varia de 0,706 a 4,3 por classe; a medida FOUT varia de 0,495 a 1,465 por classe; e a medida RFC varia de 7,091 a 43,479 por classe. Quanto às medidas de coesão, a medida LCOM varia de 49,175 a 2.360,36 por classe; LCOM2 varia de 0,465 a 1,0 por classe; LCOM4 varia de 28,298 a 1.490,186 por classe; e as medidas TCC e LCC variam de 0,028 a 0,373 por classe.

Tabela 19 Estatística descritiva para o domínio *Home & Education*

Medidas	Mínimo	Máximo	25%	Mediana	75%	Desvio Padrão
NC	30	3.367	82	294	698	648,9338
LOC	1.791	225.849	9.864	26.394	63.231	45.379,392
NOM	4,484	20,829	5,78788	7,02073	9,92865	3,299765
NOA	1,594	16,843	3,60465	4,36768	6,8	3,049932
CC	14,42	241,507	25,8719	33,3166	57,747	45,28664
WMC	3,515	83,279	13,1333	17,1765	31,8338	18,1105
DIT	0	5,947	1,09155	1,47899	2,16206	1,019105
NOC	0	6,067	0,38931	0,60027	0,98749	1,087493
CBO	0,706	4,3	1,08808	1,39935	1,81395	0,722481
FOUT	0,495	1,465	0,82353	0,88857	0,98039	0,170905
RFC	7,091	43,479	10,4091	12,6667	19,1463	7,582734
LCOM	49,175	2.360,36	124,418	379,568	605,426	559,8976
LCOM2	0,465	1	0,78048	0,87224	0,93561	0,139744
LCOM4	28,298	1.490,18	72	191,693	385,351	327,8948
TCC	0,028	0,373	0,06071	0,102	0,15098	0,073181
LCC	0,028	0,373	0,06071	0,102	0,15098	0,073181

Para os sistemas do domínio *Science & Engineering* (SE), a estatística descritiva é apresentada na Tabela 20. Em relação ao tamanho, os sistemas do domínio SE tendem a ter mínimo de 3,0 e máximo de 4.576 classes. O quartil inferior (1/4 da amostra) está entre três a 74,5 classes. O quartil superior (3/4 da amostra) está entre 560 a 4.576 classes. Esses sistemas possuem de 119 a 338.401 linhas de código. Além disso, a quantidade de métodos varia de 3,786 a 19,335 métodos por classe e a quantidade de atributos varia de 1,082 a 15,844 atributos por classe. Quanto às medidas de complexidade, a medida CC varia de 8,333 a 135,041 por classe. A medida WMC varia de 4,333 a 61,193 por classe.

Tabela 20 Estatística descritiva para o domínio *Science & Engineering*

Medidas	Mínimo	Máximo	25%	Mediana	75%	Desvio Padrão
NC	3,0	4.576	74,5	210	560	745,6167
LOC	119	338.401	5.764,5	20.354	50.633,5	70.042,315
NOM	3,786	19,335	5,67424	7,30435	9,761	3,333921
NOA	1,082	15,844	2,62395	3,65714	5,26278	2,932226
CC	8,333	135,041	20,8606	29,8712	58,0445	29,752079
WMC	4,333	61,193	9,93189	16,0653	24,7604	14,184674
DIT	0	3,333	1,42655	1,6667	1,93757	0,604229
NOC	0	10,13	0,41879	0,7	1,37719	1,558616
CBO	0,767	5,702	1,05325	1,30435	1,63627	0,875592
FOUT	0,508	1,303	0,75893	0,84127	0,91922	0,131414
RFC	5,548	78,449	7,98126	14,6117	21,3073	14,271983
LCOM	6,333	2.709,83	76,7363	184,326	600,059	600,61089
LCOM2	0,514	0,998	0,73547	0,84326	0,90913	0,120157
LCOM4	7,0	1.165,46	41,2645	127,246	342,341	318,65253
TCC	0,019	0,465	0,03942	0,08207	0,1323	0,089580
LCC	0,019	0,465	0,03942	0,08207	0,1323	0,089580

Em relação às medidas de herança, a medida DIT varia de 0,0 a 3,333 por classe (em que a maior hierarquia entre classes é de aproximadamente 3,0). A medida NOC varia de zero a 10,13 por classe, ou seja, a menor quantidade de subtipos em uma classe é 0,0 e a maior quantidade de subtipos em uma classe é

de aproximadamente 10. Quanto às medidas de acoplamento, a medida CBO varia de 0,767 a 5,702 por classe; a medida FOUT varia de 0,508 a 1,303 por classe; e a medida RFC varia de 5,548 a 78,449 por classe. Quanto às medidas de coesão, a medida LCOM varia de 6,333 a 2.709,826 por classe; LCOM2 varia de 0,514 a 0,998 por classe; LCOM4 varia de 7,0 a 1165,464 por classe; e as medidas TCC e LCC variam de 0,019 a 0,465 por classe.

Para os sistemas do domínio *Security & Utilities* (SU), a estatística descritiva é apresentada na Tabela 21. Em relação ao tamanho, os sistemas do domínio SU tendem a ter mínimo de 5,0 e máximo de 3.290 classes. O quartil inferior (1/4 da amostra) está entre 5,0 a 33,25 classes. O quartil superior (3/4 da amostra) está entre 378 a 3.290 classes. Esses sistemas possuem de 602 a 330.025 linhas de código. Além disso, a quantidade de métodos varia de 3,419 a 23,206 métodos por classe e a quantidade de atributos varia de 0,025 a 18,2 atributos por classe. Quanto às medidas de complexidade, a medida CC varia de 9,156 a 226,837 por classe. A medida WMC varia de 5,25 a 75,55 por classe.

Tabela 21 Estatísticas descritivas para o domínio *Security & Utilities*

Medidas	Mínimo	Máximo	25%	Mediana	75%	Desvio Padrão
NC	5,0	3290	33,25	84	378	641,2042
LOC	602	330025	2809,25	15332,5	46414	64567,8249
NOM	3,419	23,206	5,34737	7,01918	9,75043	3,764335
NOA	0,025	18,2	3,54357	5,53408	6,87207	3,218026
CC	9,156	226,837	32,7793	45,26417	60,6913	39,494206
WMC	5,25	75,55	12,3313	19,56451	25,4295	15,546969
DIT	0	4,546	1,20053	1,47485	2,14779	0,925275
NOC	0	2,397	0,4135	0,32167	0,6865	0,515371
CBO	0,833	2,8	1,24617	1,37422	1,72381	0,473742
FOUT	0,25	1,0	0,71374	0,87338	0,95409	0,87338
RFC	6,267	53,696	11,5109	14,60402	19,4722	8,627889
LCOM	9,8	2438,17	32,0054	150,24477	430,870	573,034053
LCOM2	0,221	0,989	0,67953	0,86988	0,93616	0,181744

“Tabela 21, conclusão”

Medidas	Mínimo	Máximo	25%	Mediana	75%	Desvio Padrão
LCOM4	7,786	8957,14	28,1991	98,94586	295,792	320,6688
TCC	0	0,34	0,5662	0,07963	0,11855	0,073293
LCC	0	0,455	0,5662	0,07963	0,11855	0,226548

Em relação às medidas de herança, a medida DIT varia de 0,0 a 4,546 por classe (em que a maior hierarquia entre classes é de aproximadamente 5,0). A medida NOC varia de 0,0 a 2,397 por classe, ou seja, a menor quantidade de subtipos em uma classe é 0,0 e a maior quantidade de subtipos em uma classe é de aproximadamente 2,0. Quanto às medidas de acoplamento, a medida CBO varia de 0,833 a 2,8 por classe; a medida FOUT varia de 0,25 a 1,0 por classe; e a medida RFC varia de 6,267 e 53,696 por classe. Quanto às medidas de coesão, a medida LCOM varia de 9,8 a 2.438,173 por classe; LCOM2 varia de 0,221 a 0,989 por classe; LCOM4 varia de 7,786 a 8.957,135 por classe; a medida TCC varia de 0,0 a 0,34 por classe; e a medida LCC varia de 0,0 a 0,455 por classe.

Para os sistemas do domínio *System Administration* (SA), a estatística descritiva é apresentada na Tabela 22. Em relação ao tamanho, os sistemas do domínio SA tendem a ter mínimo de 2,0 e máximo de 2.445 classes. O quartil inferior (1/4 da amostra) está entre 2,0 a 16,5 classes. O quartil superior (3/4 da amostra) está entre 264 a 2.445 classes. Esses sistemas possuem de 245 a 83.312 linhas de código.

Tabela 22 Estatística descritiva para o domínio *System Administration*

Medidas	Mínimo	Máximo	25%	Mediana	75%	Desvio Padrão
NC	2,0	2.445	16,5	93	264	450,6662
LOC	29	83.312	1.013,5	10.171	25.206	19.445,1315
NOM	1,571	17,415	5,0	6,6	8,18328	2,847474
NOA	0	11,345	2,30348	3,2507	5,78889	2,42226
CC	0	169,389	15,963	28,10569	42,6847	31,152547
WMC	0	52,806	6,41189	10,4	15,98438	8,628082
DIT	0	3,546	1,10948	1,28485	1,65654	0,714175
NOC	0	4,0	0,22794	0,52332	0,8102	0,665603
CBO	0,769	5,683	1,0	1,14286	1,4016	0,801819
FOUT	0,533	1,1	0,8	0,89427	0,97374	0,119127
RFC	2,571	46,25	7,85193	11,125	17,20773	9,286549
LCOM	0	2.056,061	16,22281	125,64423	353,5285	366,992141
LCOM2	0	0,997	0,75037	0,85154	0,93002	0,185114
LCOM4	0	674,535	14,75214	67,04545	208,68851	150,261207
TCC	0	0,478	0,06496	0,10046	0,19431	0,115261
LCC	0	0,478	0,06496	0,10046	0,19431	0,115261

Além disso, a quantidade de métodos varia de 1,571 a 17,415 métodos por classe e a quantidade de atributos varia de 0,0 a 11,345 atributos por classe. Em relação às medidas de herança, a medida DIT varia de 0,0 a 3,546 por classe (em que a maior hierarquia entre classes é de aproximadamente 4,0). A medida NOC varia 0,0 a 4,0 por classe, ou seja, a menor quantidade de subtipos em uma classe é 0,0 e a maior quantidade de subtipos em uma classe é 4,0.

Quanto às medidas de acoplamento, a medida CBO varia de 0,769 a 5,683 por classe; a medida FOUT varia 0,533 a 1,1 por classe; e a medida RFC varia de 2,571 e 46,25 por classe. Quanto às medidas de coesão, a medida LCOM varia de 0,0 a 2.056,061 por classe; LCOM2 varia de 0,0 a 0,997 por classe; LCOM4 varia de 0,0 a 674,535 por classe; e as medidas TCC e LCC variam de 0,0 a 0,478 por classe.

7.4 Discussão

Em relação aos domínios analisados, os domínios ***D***, ***BE*** e ***G*** tendem a ter sistemas com maior quantidade de classes, sendo os maiores representantes com 8.040, 5.290, 3.072 classes, respectivamente. Além disso, 25% dos valores mais elevados dos sistemas (3º quartil) estão entre [779,75; 8.040], [1.420,5; 5.290] e [419,25; 3.072] classes, respectivamente. Estes três domínios também possuem sistemas com grande quantidade de linhas de código. Por exemplo, para o domínio ***D***, há um projeto com **LOC** = 430.124. Para o domínio ***G***, há um projeto com **LOC** = 3.644.005.

Na literatura, há pesquisas que ressaltam que, geralmente, os sistemas empresariais, de negócio, jogos e desenvolvimento (software para o sistema) tendem a ser customizáveis e/ou específicos, além de ter arquitetura com módulos específicos (AMPATZOGLOU; CHATZIGEORGIOU, 2007; KUMAR; MAHESHWARI; KUMAR, 2003; NAH; FAA; CATA, 2001; PRESSMAN, 2014; SHI; ZHAO, 2009). Dessa forma, essas características das medidas de tamanho podem ter relação com as características descritas nesses estudos.

Outro ponto a ser observado é em relação à modularidade: os domínios ***BE*** e ***D*** têm relação inversa para esse atributo. O domínio **BE** tende a ter sistemas menos coesos, assim como os domínio ***HE*** e ***GPH***. Por exemplo, o maior valor das medidas **TCC** e **LCC** nas classes dos sistemas de software do domínio ***BE*** é 0,3. Uma classe com valor dessas medidas <= 0,5 pode ser considerada de baixa coesão (BIEMAN; KANG 1995). Por outro lado, o domínio ***D*** tende a ter sistemas mais coesos (no 3º quartil, há classes nos sistemas de software analisados com valores entre 0,4 e 0,8), assim como o domínio ***AV***, que também possui sistemas de software com valores para a medida **TCC** e **LCC** maiores que 0,5 (**TCC** = **LCC** = 0,903).

Além disso, os sistemas do domínio ***BE*** tem acoplamento maior entre pacotes, reforçando as características discutidas no parágrafo anterior. Cabe ressaltar que o aumento do acoplamento implica no aumento das dependências interclasses, tornando o código menos modular e menos adequado para reutilização (AL DALLAH, 2013). Em outras palavras, para reutilizar funções e módulos em um sistema acoplado, o desenvolvedor tende a incluir código com funções importantes para a funcionalidade central da classe. O resultado dessa investigação inicial indica que em média sistemas do domínio ***D*** tendem a ser mais fáceis de manter e reutilizar. Para um sistema de alta qualidade interna, o ideal é ter comportamento similar ao encontrado nesse domínio.

Quanto à herança, evidências de estruturas longas de herança podem ser observadas nos domínios ***HE*** e ***G***. Nesses domínios, há casos em que o valor da medida **DIT** aproximou de 6,0, ou seja, caminho máximo de herança de uma classe para a classe raiz é, em média, de até 6,0 classes. Em teoria, quanto mais profunda uma classe se posiciona na hierarquia do sistema de software, mais métodos e variáveis provavelmente herdará (AL DALLAH, 2013). Esse arranjo pode tornar a classe mais complexa (AL DALLAH, 2013). O valor recomendado da medida **DIT** em única classe é inferior ou igual a 2,0 (FERREIRA et al., 2012).

Em relação à medida **NOC**, a maior quantidade de subtipos encontrados em uma classe foi registrado nos domínios ***BE*** e ***SE***, com média de 61 e 10,13 subtipos, respectivamente. Esse resultado indica que, nesses domínios, há classes com alta reutilização e/ou abstração imprópria da classe pai. Nesse caso, os desenvolvedores desses tipos de sistema devem estar atentos e verificar se há necessidade de agrupamento dessas classes relacionadas, para reduzir a complexidade e a dependência entre elas.

Os resultados da análise descritiva indicam o domínio em que um sistema de software pertence pode ter impacto no valor das medidas de software

em suas diferentes propriedades: tamanho, complexidade, herança, coesão e acoplamento. Cada uma das análises realizadas contribuiu com uma visão específica de cada domínio: diferenças entre o nível médio dessas propriedades puderam ser observadas. Além disso, alguns domínios possuem similaridades e dissimilaridades sob o ponto de vista de acoplamento, coesão, herança, tamanho e complexidade. Por exemplo, em relação à coesão, os domínios ***D*** e ***AV*** tendem a ser mais coesos e a variação das medidas possui valores próximos. A amplitude das medidas **TCC** e **LCC** (diferença entre o valor máximo e mínimo) é, respectivamente, 0,59 e 0,529.

Em um estudo prévio (SANTOS et al., 2014), foi realizado um experimento para encontrar similaridades entre as estruturas de sistemas de software de diferentes domínios por meio de medidas. Os testes estatísticos foram realizados em uma amostra de 150 sistemas, classificados em macro-categorias e micro-categorias (composta dos 10 domínios do Sourceforge) para identificar quais dessas classificações proporcionam melhor informação sobre os sistemas. Utilizando a Análise de *Cluster*, os resultados do estudo indicam que alguns domínios possuem características similares em relação à complexidade, ao tamanho, a modularização e a abstração/estabilidade, como os domínios ***D***, ***AV*** e ***C***. Nesse caso, esses domínios com características similares e dissimilares devem ser tratados com cuidado, quando se considera práticas para análise e melhoria da qualidade.

7.5 Considerações finais

No estudo proposto nesta seção, tem-se que, em relação às 16 medidas de qualidade interna estudadas, é possível identificar indícios de similaridade entre sistemas de software de diferentes domínios. Entretanto, cabe ressaltar duas questões:

a) A abordagem apresentada não é capaz de explicar explicitamente a similaridade, que pode ser justificada por diversos fatores, por exemplo, os processos de desenvolvimento de software similares, as práticas e os princípios de desenvolvimento de sistemas de software similares e ter a mesma equipe de desenvolvimento. Para que a influência desses fatores seja confirmada, é necessário a condução de um estudo qualitativo no código dos sistemas em cada grupo;
b) Com a quantidade de medidas e a abordagem utilizada, pode-se afirmar que os sistemas listados nas tabelas de cada domínio possuem similaridades e dissimilaridades sob o ponto de vista do acoplamento, da coesão, da herança, do tamanho e da complexidade. Dessa forma, essas medidas, se teoricamente correlacionadas, podem fornecer informações sobre a qualidade de sistemas de software OO desenvolvidos em Java.

8 ANÁLISE FATORIAL EXPLORATÓRIA - ESTUDO DA CORRELAÇÃO ENTRE MEDIDAS DE QUALIDADE INTERNA DE SOFTWARE

8.1 Considerações iniciais

A Análise Fatorial Exploratória (AFE) é uma técnica de análise de dados cujo objetivo é descobrir e analisar a estrutura de um conjunto de variáveis interrelacionadas (MAROCO, 2010). Essas interrelações são expressas em uma escala de medida para fatores que, de alguma forma, mais ou menos explícita, controlam e explicam as variáveis originais (HAIR et al., 2009; MAROCO, 2010). Dessa forma, se duas ou mais variáveis estão correlacionadas (e a correlação não é hipotética), essa associação resulta da partilha de uma característica comum não diretamente observável (fator latente comum).

O objetivo primordial da AFE é atribuir *scores* (quantificações) a construtos ou a fatores não diretamente observáveis (MAROCO, 2010). Por exemplo, para medir "fatores que influenciam qualidade de sistemas de software" um investigador pode definir vários itens que meçam "a quantidade de linhas de código", "a quantidade de classes acopladas", "o tamanho da hierarquia entre classes" etc. O *score* da qualidade de software poderia ser dado pelo somatório de valores atribuídos, entretanto, na AFE, um *score* é produzido ponderando as respostas altamente correlacionadas (MAROCO, 2010). Esse novo *score* é uma representação da informação presente nas diferentes variáveis e é capaz de resumir a informação presente em muitas variáveis, em uma quantidade reduzida de fatores (HAIR et al., 2009; MAROCO, 2010). Neste capítulo, os fatores que influenciam a qualidade de software em relação à característica Manutenibilidade são estudados, utilizando medidas de software

como parâmetros, que representam propriedades como tamanho, complexidade, acoplamento, coesão e herança.

Esta seção está organizada da seguinte forma. As variáveis dependentes e independentes do estudo são descritas na subseção 8.2. Os resultados da análise de validade e confiabilidade da AFE são discutidos na subseção 8.3. Os resultados da AFE são mostrados na subseção 8.4. A discussão dos resultados encontrados é apresentada na subseção 8.5.

8.2 Descrição das variáveis

Considerando o modelo de qualidade do produto, suas características e subcaracterísticas, esta pesquisa baseou-se nos seguintes grupos de variáveis (AL DALLAL, 2013; INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 2011):

a) **Qualidade Externa - Manutenibilidade**: a manutenibilidade de um sistema de software OO depende de vários elementos, por exemplo, do próprio sistema (e seu conjunto de classes), da experiência da equipe de desenvolvimento e do tempo de existência. Sistemas com classes com baixa manutenibilidade devem ser cuidadosamente testados para reduzir a propensão em falhas e aprimorar a legibilidade, facilitando futuras manutenções;
b) **Qualidade Interna - Medidas de Software**: os atributos de qualidade interna, por exemplo, o tamanho, a coesão, o acoplamento, a herança e a complexidade, podem ser mensurados por meio de informações obtidas em classes de sistemas de software. O tamanho e a complexidade são considerados atributos chave para a manutenibilidade de sistemas, pois influenciam na propensão em

falhas e na reusabilidade do sistema. Acoplamento e coesão são propriedades que se referem à extensão em que as classes do sistema estão relacionadas. A modularidade do sistema influencia sua legibilidade, sua reusabilidade e sua manutenibilidade.

8.3 Validade e confiabilidade

Após a remontagem da base de dados, a validade de construtos foi verificada. A adequação dos dados e da solução encontrada pela AFE foi validada por meio do índice de *Kaiser-Meyer-Olkin* (KMO) e da execução do teste de esfericidade de *Bartlett* para os dados gerados na matriz fatorial.

Com o teste de esfericidade de *Bartlett*, foi avaliada a presença de correlações entre as variáveis (medidas de software). Com o índice de KMO, o objetivo foi quantificar o grau de intercorrelação entre essas variáveis, definindo a qualidade da amostra e a adequação da AFE (HAIR et al., 2009; TREIBLMAIER; FILZMOSER, 2010). As informações de validação da AFE são apresentadas na Tabela 23. Com o índice de KMO = 0.718, a análise com recomendação média pode ser utilizada (HAIR et al., 2009; MAROCO, 2010). Em relação ao teste de esfericidade, significância inferior a 0,001 indica que a matriz de correlação encontrada não é uma identidade, sendo adequada para a aplicação da AFE.

Tabela 23 Resultado do Índice KMO e Teste de Esfericidade de *Bartlett*

Teste	**Índice**	**Valores**
Medida de adequação da amostra	KMO (*Kaiser-Meyer-Olkin)*	0.718
Teste de esfericidade de *Bartlett*	Aproximação $\chi 2$	5993,497
	df (grau de liberdade)	105
	p (significância)	0,0

8.4 Resultados

Em relação à quantidade de fatores extraídos, pode-se observar na Tabela 24, uma estrutura de cinco fatores que explica 67% da variância dos dados. O método de extração de fatores utilizado foi a Fatoração Alfa com o método de rotação *Varimax*. A matriz de fatores iniciais, que indica a relação entre as variáveis estudadas, raramente resulta em fatores "limpos", ou seja, fatores que possuem potencialmente cargas cruzadas (repetidas em dois ou mais fatores), e não maximizadas em cada fator. Os métodos de rotação da matriz transformam a matriz de fatores em uma matriz rotacionada, maximizada, significativa, mais simples e mais fácil de interpretar. Com o giro de fatores, podem ser identificados fatores que possuam variáveis com alta correlação e outros com variáveis com baixa correlação (HAIR et al., 2009).

Para especificar as variáveis cujas cargas fatoriais têm significância prática, o tamanho da amostra e a quantidade de variáveis devem ser considerados. A significância prática na escolha das cargas fatoriais é utilizada para fazer inferências preliminares da matriz fatorial. Em síntese, o método considera que as cargas maiores que 0,30 atingem o nível mínimo aceitável, cargas entre 0,40 e 0,50 são consideradas mais importantes e cargas maiores que 0,50 são consideradas com significância prática (HAIR et al., 2009). Por exemplo, em uma amostra de 350 ou mais casos, cargas fatoriais maiores que 0,30 podem ser consideradas significantes (HAIR et al., 2009). Seguidas as recomendações, a primeira matriz de fatores rotacionada extraída, é apresentada na Tabela 24, em que as cargas significantes são grafadas em negrito.

Tabela 24 Fatores iniciais extraídos

Medidas	Fator 1	Fator 2	Fator 3	Fator 4	Fator 5	Comunalidades
LOC	**0,929**	0,014	0,038	-0,004	0,082	**0,872**
NOM	**0,783**	0,168	-0,144	0,294	-0,193	**0,786**
NOA	**0,754**	-0,012	0,115	-0,055	0,146	**0,607**
CC	**0,884**	-0,018	0,077	-0,054	0,127	**0,807**
WMC	**0,703**	-0,066	0,029	0,205	0,153	**0,566**
DIT	0,134	0,004	0,104	**0,343**	**0,455**	**0,354**
NOC	0,011	0,078	-0,082	-0,031	0,180	**0,046**
CBO	**0,526**	0,044	-0,063	**0,422**	-0,029	**0,462**
FOUT	0,117	-0,039	0,092	**0,910**	-0,019	**0,853**
RFC	**0,708**	0,170	-0,185	0,179	-0,126	**0,613**
LCOM	0,003	**0,967**	-0,014	0,076	0,118	**0,955**
LCOM2	0,064	0,144	0,144	**0,669**	0,076	**0,499**
LCOM4	0,097	**0,954**	0,017	0,059	0,079	**0,929**
TCC	-0,005	0,010	**0,938**	0,116	-0,062	**0,897**
LCC	-0,001	-0,005	**0,881**	0,111	-0,044	**0,790**
Scores	**0,945**	**0,967**	**0,924**	**0,899**	**0,461**	**-**

Na solução fatorial rotacionada (Tabela 24), cada variável tem cargas significantes (definidas como um valor acima de 0,30), entretanto há duas variáveis que cruzam sobre dois fatores: as medidas **CBO** e **DIT**. Para a medida **CBO**, as cargas **0,501** (**Fator 1**) e **0,422** (**Fator 4**), não são substancialmente representativas para os fatores. O mesmo pode ser visto para a medida **DIT**, com as cargas **0,343** (**Fator 4**) e **0,455** (**Fator 5**). Nesse caso, sugere-se que o modelo fatorial seja reespecificado, com opções de ajustes, por exemplo, eliminando as variáveis que se cruzam, mudando os métodos de rotação, reduzindo/aumentando a quantidade de fatores e modificando o tipo de extração de fatores (HAIR et al., 2009; JOHNSON; WICHERN, 2007). Além disso, os valores de comunalidade abaixo de 0,50 indicam que a variável em questão não se ajusta à estrutura definida pelas outras variáveis (HAIR et al., 2009).

Dadas as recomendações, as variáveis (medidas) **CBO**, **NOC** e **DIT** foram eliminadas e o método de Componentes Principais foi utilizado para

extrair os fatores na nova análise. Nem todos os fatores são aproveitáveis em uma análise fatorial exploratória A, determinação da quantidade de fatores pode ser facilitada por meio da análise do gráfico de *screen plot* ou gráfico do teste de autovalor. O autovalor é calculado pela soma dos quadrados das cargas de cada variável representada pelo fator obtido (HAIR et al., 2009).

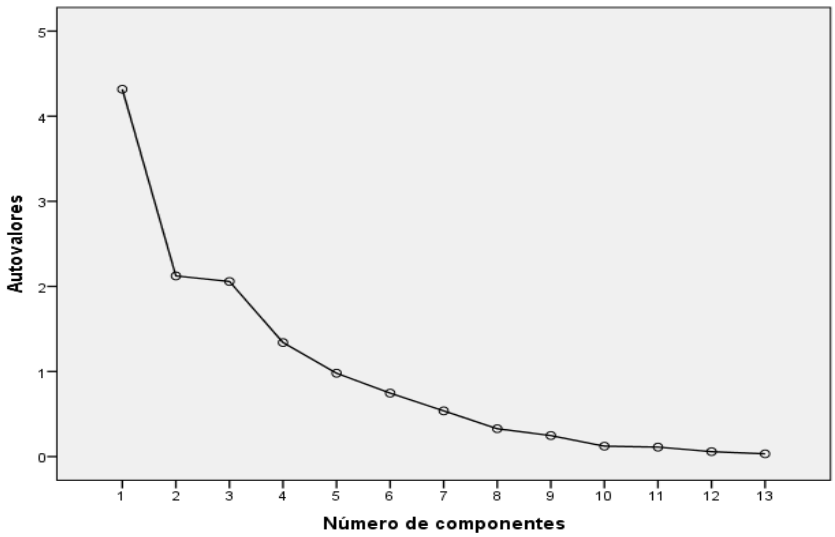


Figura 19 Teste do autovalor para a análise

Pelo critério do autovalor (*eigenvalue*, ou autovalor, maior que 1,0), deve-se manter quatro fatores (Figura 19). Em relação à quantidade de fatores extraídos, a estrutura de quatro fatores explica **82%** da variância dos dados. Na Tabela 25, tem-se o modelo reespecificado com uma solução de quatro fatores cujo **KMO = 0,710**. Além da variância comum (variância que uma variável divide com outras variáveis medidas), tem-se a variância única específica para uma variável. A comunalidade é a proporção de variância comum presente em uma variável e representa a proporção da variância explicada pelos fatores extraídos (HAIR et al., 2009). Para considerar uma variável significativa para o

modelo fatorial, recomenda-se que comunalidades devam exceder o valor de 0,60 para as variáveis em pesquisa (HAIR et al., 2009), conforme visualizado na Tabela 25. A interpretação dos fatores prossegue com o exame da matriz rotacionada, da descrição e da caracterização com base nas cargas fatoriais significantes. De forma análoga à Tabela 24, apenas as cargas significantes são mostradas.

Tabela 25 Fatores extraídos após reespecificação do modelo fatorial

Medidas	Fator 1	Fator 2	Fator 3	Fator 4	Comunalidades
LOC	**0,937**	0,034	0,017	-0,023	**0,880**
NOM	**0,788**	0,147	-0,167	0,304	**0,762**
NOA	**0,818**	-0,001	0,143	-0,093	**0,698**
CC	**0,913**	0,004	0,062	-0,071	**0,842**
WMC	**0,750**	-0,081	0,053	0,218	**0,620**
FOUT	0,119	-0,033	0,087	**0,899**	**0,831**
RFC	**0,744**	0,153	-0,214	0,189	**0,658**
LCOM	0,007	**0,983**	-0,016	0,070	**0,972**
LCOM2	0,060	0,157	0,132	**0,849**	**0,766**
LCOM4	0,095	**0,981**	0,014	0,054	**0,975**
TCC	-0,003	0,006	**0,950**	0,113	**0,916**
LCC	0,002	-0,006	**0,949**	0,105	**0,911**

Fator 1 - Redução da Complexidade (LOC, NOM, NOA, CC, WMC e RFC). Formas comuns de acoplamento ocorrem com (i) variáveis de instância, (ii) variáveis locais de métodos ou de seus parâmetros, (iii) chamada de serviços em outra classe, (iv) uma classe derivada direta ou indiretamente de outra ou (v) uma interface implementa por uma classe. A medida **RFC** corresponde à quantidade de métodos distintos e de construtores invocados por uma classe. Esses métodos incluem os da classe, os herdados e os que podem ser chamados em outros objetos. Se o valor da medida **RFC** para uma classe for grande, isso significa que existe elevada complexidade. Por exemplo, uma chamada de método na classe pode resultar em grande quantidade de diferentes chamadas de um método e de outras classes. A complexidade ciclomática corresponde à

complexidade relativa de um método em relação a outro. No entanto, é possível afirmar que um método com complexidade ciclomática menor, normalmente, é menos complexo. Como as medidas de tamanho, de complexidade e de acoplamento crescem na mesma direção, isso significa que se a quantidade de métodos, de atributos e de classes (em termos de linhas de código) é grande, a quantidade de métodos invocados tende a ser grande, aumentando a complexidade do código. Esse fator representa a necessidade de manter a complexidade reduzida, equilibrando os valores para as medidas que compõem o fator.

Fator 2 - Coesão não Normalizada (LCOM e LCOM4). As medidas **LCOM** e **LCOM4** correspondem à quantidade de pares de métodos de classes que não utilizam atributos em comum. Essas medidas visam detectar classes problemáticas. Valor alto das medidas **LCOM** e **LCOM4** indica baixa coesão; valor baixo das medidas **LCOM** e **LCOM4** indica alta coesão. Entretanto, essas medidas não são normalizadas, ou seja, não têm limites superiores. Além disso, a medida **LCOM4** é uma variação da medida **LCOM**, que adicionalmente representa invocações de métodos que utilizam o mesmo atributo. Esse fator representa a necessidade de aumentar a coesão das classes do sistema, mantendo equilibrada a utilização de atributos em comum, de métodos de uma mesma classe.

Fator 3 - Coesão Normalizada (TCC e LCC). As medidas **TCC** e **LCC** são de coesão normalizadas (possuem limite superior e limite inferior [0..1]). Essas medidas indicam a diferença entre uma "boa coesão" e a "má coesão", em que valores próximos de 1 indicam alta coesão e valores próximos de 0 indicam baixa coesão. O cálculo dessas medidas é mais restrito, pois não inclui construtores e destrutores, além de ser menos influenciado pelo tamanho. Além disso, a medida **LCC** considera pares de métodos indiretamente conectados. Por considerar conexões mais complexas entre classes que as

medidas **LCOM** e **LCOM4**, esse fator representa a necessidade de reduzir as invocações entre métodos em uma mesma classe. Isso acontece, pois, mesmo que atributos não sejam diretamente utilizados, a invocação de mesmo método com esse atributo pode aumentar a complexidade da classe e sugerir mais responsabilidades que a classe deveria ter.

Fator 4 - Aumento do Grau de Modularidade (FOUT e LCOM2). Para um sistema de software tenha arquitetura madura e sustentável, o desenvolvedor deve considerar o acoplamento e a coesão desse sistema, pois, baixo acoplamento e alta coesão são importantes para sistemas modularizados. Uma classe não deve assumir responsabilidades que não são suas e, ignorando esse princípio, o sistema passa a ter dificuldades de manutenção e de reúso de suas classes e de seus pacotes. Classes com valor alto para a medida **FOUT** tendem a ter grande quantidade de métodos e de linhas de código. Classes com valor alto para a medida **LCOM2** tendem a ter mais responsabilidades que deveriam. A correlação com a medida **LCOM2** indica que ambas crescem na mesma direção, pois, quanto maior é o valor para a medida **FOUT**, maior é o valor para a medida **LCOM2**. Assim, alto acoplamento pode implicar em baixa coesão. Esse fator representa a necessidade de reduzir o percentual de métodos que não acessam um atributo específico e as dependências entre classes. Dessa forma, o sistema aumenta seu grau de modularidade.

8.5 Discussão

Os fatores extraídos descrevem combinações de medidas software cuja característica, na prática, pode promover melhorias no código. Essas medidas estão correlacionadas com a qualidade interna de sistemas de software OO. A partir desses resultados, pode-se construir um cenário de utilização dessas

medidas como apoio à melhoria da qualidade interna de sistemas de software com foco na manutenibilidade.

Os sistemas de software devem ter código menos complexo; caso contrário, dificulta o processo de desenvolvimento: um único desenvolvedor pode não ser capaz de entender o sistema como um todo (código incompreensível, pouco legível e mal documentado). Além disso, o sistema pode ser difícil de ser utilizado e, consequentemente, difícil de manter e evoluir. Dessa forma, a redução da complexidade (**Fator 1**) é o fator que representa os esforços que devem ser feitos para manter o código menos complexo. Para garantir a qualidade interna do software, é necessário rever as invocações de métodos, a quantidade de atributos utilizados, a quantidade de linhas de código em um único método e manter a quantidade de caminhos executados o menor possível (instruções como *if/else*, *for*, *while*, *try/catch* e expressões lógicas). As medidas que compõem o fator (**LOC, NOM, NOA, CC, WMC** e **RFC**) tendem a serem melhores indicadoras dessa característica.

Além de menos complexo, os sistemas de software devem ter um código coeso. A coesão refere-se ao nível com os quais diferentes componentes de um software estão interrelacionados uns com os outros. A coesão pode ser calculada de duas maneiras: por medidas não normalizadas (**Fator 2**) e por medidas normalizadas (**Fator 3**). O **Fator 2** indica a avaliação da quantidade de pares de métodos entre classes que não utilizam atributos em comum. Além disso, o **Fator 2** sugere a avaliação da quantidade de invocações de métodos em uma classe que utilizam o mesmo atributo. Dessa forma, é possível detectar as classes problemáticas e determinar melhor suas responsabilidades, facilitando a legibilidade e manutenção do código. As medidas **LCOM** e **LCOM4** compõem esse fator e tendem a ser melhores indicadoras dessa característica. O **Fator 3** indica se as classes do sistema possuem "boa coesão" ou "má coesão". O cálculo da coesão nesse fator considera pares de métodos direta e indiretamente

conectados, em que, para ter boa coesão, é necessário reduzir as invocações entre métodos em uma mesma classe. As medidas **TCC** e **LCC** compõem esse fator e tendem a ser melhores indicadoras dessa característica.

Por fim, para ter qualidade, os sistemas de software devem ser modulares. Com a modularização, é possível preservar as funções em partes específicas do código, permitindo que sejam utilizadas em toda parte do sistema, com baixo grau de dependência entre essas partes. Um baixo acoplamento e uma alta coesão são importantes para sistemas modularizados. Uma classe não deve assumir responsabilidades que não são suas e deve evitar o excesso de dependência entre pacotes. O aumento do grau de modularidade (**Fator 4**) do sistema é consequência do processo de modificações visando melhorar a estrutura interna do código quanto a sua coesão e acoplamento. As medidas **FOUT** e **LCOM2** compõem esse fator e tendem a ser melhores indicadoras dessas características. Dessa forma, um sistema altamente modular, coeso e de baixa complexidade tende a ser altamente manutenível e reutilizável.

8.6 Considerações finais

Em um processo de desenvolvimento de software, é preciso medir a qualidade não só do produto final, mas dos produtos intermediários gerados durante o processo. Com a utilização de um sistema de coleta de valores de medidas, os desenvolvedores, os gestores de qualidade e os mantenedores poderão avaliar melhor a sua produtividade e adaptabilidade do código às necessidades dos *stakeholders* envolvidos. Com isso, pode-se ter redução de tempo para executar tarefa de manutenção do sistema.

No estudo proposto nesta seção, tem-se que, em relação às 15 medidas de qualidade interna implementadas no *plug-in O3SMeasures*, podem ser identificados fatores que caracterizam a qualidade interna de sistemas de

software OO, considerando diferentes domínios. Entretanto, cabe ressaltar duas questões:

a) A tendência identificada nessa análise é **Redução da Complexidade**, **Coesão Normalizada**, **Coesão não Normalizada** e **Aumento do Grau de Modularidade** são fatores a serem considerados por desenvolvedores para que um sistema seja manutenível;
b) De forma específica, as medidas que compõem os fatores estão correlacionadas, podendo ser indicadoras de aspectos de qualidade, como acoplamento, coesão e complexidade, importantes na caracterização da qualidade interna de sistemas de software OO.

9 MODELO DE EQUAÇÕES ESTRUTURAIS UTILIZANDO O PLS-SEM

9.1 Considerações iniciais

A técnica PLS-SEM tem sido amplamente utilizada pela academia para explicar a variabilidade das variáveis dependentes em um modelo (CHIN; DIBBERN, 2009; LEE; XIA, 2010; LOWRY; GASKIN, 2014; URBACH; AHLEMANN, 2010). No PLS-SEM o objetivo é verificar as relações entre variáveis, por meio de suposições teóricas predefinidas. No modelo, são analisadas as estimativas reais dos indicadores e dos construtos estudados e, com nível aceitável de ajuste do modelo, a interpretação das variáveis estatísticas revela a natureza dessa relação multivariada.

Nesta seção, é apresentado um modelo SEM com objetivo de caracterizar a qualidade interna de software em sistemas Java *open-source* quanto à Manutenibilidade. Uma das características básicas de um modelo SEM é testar uma teoria de ordem causal entre um conjunto de variáveis. Neste estudo, por exemplo, a teoria propõe que o relacionamento entre medidas de software em sistemas OO desenvolvidos em Java influencia a qualidade interna do software sob a Manutenibilidade. É possível verificar tal relação causal com o uso da SEM. Essa técnica oferece ao pesquisador a possibilidade de investigar quão bem as variáveis preditoras (**Coesão Normalizada**, **Coesão não Normalizada** e **Redução da Complexidade**) explicam a variável dependente (**Aumento do Grau de Modularidade**) e, também, qual das variáveis preditoras é a mais importante.

Esta seção está organizado da seguinte forma. O modelo de pesquisa proposto e as hipóteses são descritas na subseção 9.2. A metodologia do experimento é apresentada na subseção 9.3. Os resultados do PLS-SEM são

descritos na subseção 9.4. As discussões e as conclusões são apresentadas na subseção 9.5.

9.2 Modelo de pesquisa e as hipóteses

A manutenção é uma etapa fundamental no ciclo de vida do desenvolvimento de um software e começa quando o sistema de software é entregue (AL DALLAH, 2013). A manutenibilidade de um software pode ser definida como a facilidade de um software ser modificado, a fim de corrigir defeitos, adequar-se a novos requisitos, aumentar a suportabilidade ou adequar-se a um ambiente novo (INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 2011). Pesquisas em software *open-source* (OS) indicam que o processo de desenvolvimento do software pode influenciar na sua manutenibilidade (MIDHA; BHATTACHERJEE, 2012; SAINI; KAUR, 2014; SCACCHI, 2002; STEINMACHER et al., 2014; ZHONG; YANG; KEUNG, 2012). Por exemplo, os desenvolvedores de software, ao se vincularem em novos projetos OS, precisam aprender sobre as técnicas e os padrões utilizados nesses projetos por conta própria (SCACCHI, 2002; STEINMACHER et al., 2014). Além disso, há relatos de que sistemas de software OS são frequentemente desenvolvidos de forma diferente dos sistemas proprietários (MIDHA; BHATTACHERJEE, 2012; ZHONG; YANG; KEUNG, 2012). Entretanto, a maioria dos processos de desenvolvimento de software OS propostos é baseado nas experiências e nos requisitos específicos de cada projeto (SAINI; KAUR, 2014). Dessa forma, a qualidade do código produzido deve ser estudada e avaliada para garantir que esses sistemas sejam manuteníveis e evoluídos para o atendimento das necessidades de seus usuários.

A qualidade de um software pode ser descrita e avaliada a partir de características ou atributos de qualidade internos e externos

(INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 2011). Os atributos de qualidade externos são aqueles que indicam a qualidade da classe com base em fatores que não podem ser medidos por artefatos de software (AL DALLAH, 2013; INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 2011). Os atributos de qualidade internos são aqueles que podem ser medidos ao examinar diretamente o sistema de software (AL DALLAH, 2013; INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 2011). Por exemplo, a manutenibilidade de um sistema de software OO, atributo de qualidade externo, depende de muitos elementos, por exemplo, da organização e da arquitetura do código, da experiência da equipe responsável pela manutenção do sistema, da idade do sistema e do ambiente em que o sistema de software é modificado para se adaptar (AL DALLAH, 2013). Por outro lado, atributos de qualidade interna de um sistema de software, por exemplo, tamanho, complexidade, coesão e acoplamento, podem ser medidos extraindo informações de uma classe ou pacote de um sistema OO (AL DALLAH, 2013; BRIAND et al., 2000; INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 2011).

Dessa forma, um atributo de qualidade externa pode ser verificado e quantificado por atributos de qualidade interna. Porém, o conhecimento de um atributo de qualidade interna de um sistema é insuficiente para obter informações úteis sobre a qualidade interna. Por exemplo, é dificil antecipar a manutenibilidade de um sistema apenas avaliando o esforço necessário para manter a classe (tamanho da classe) (AL DALLAH, 2013; MORASCA, 2009). No entanto, há interesse dos profissionais de software em determinar quais atributos de qualidade interna, por exemplo, tamanho, complexidade, coesão e acoplamento, são melhores indicadores de atributos de qualidade externa (ex.: manutenibilidade) (BRIAND et al., 2000; BRIAND; DALY; WUEST, 1998; GYIMOTHY; FERENC; SIKET, 2005; MORASCA, 2009).

Na ISO-25010, a caracteristica de qualidade Manutenibilidade é composta por subcaracterísticas de qualidade, dentre elas, a modularidade. A modularidade é o grau em que um sistema de software é composto por componentes discretos, tal qual a mudança de um componente deve ter o menor impacto possível em outros componentes (INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 2011).

Para melhorar a modularidade de sistemas de software OO, têm sido propostas técnicas que visam à manutenção preventiva, por exemplo, a refatoração, como suporte fundamental para o aumento da coesão de uma classe, a redução da complexidade (de métodos longos e de classes longas) e a redução do acoplamento entre as classes para o aumento da modularidade (AL DALLAH, 2013; ERDIL et al., 2003; NAPOLI; PAPPALARDO; TRAMONTANA, 2013). Dessa forma, a utilização de medidas, em que cada uma pode avaliar uma faceta diferente do sistema de software, pode dar ao profissional de software melhor compreensão das características de qualidade. Na Figura 20, o relacionamento desses construtos é representado e como as hipóteses de pesquisa foram formuladas.

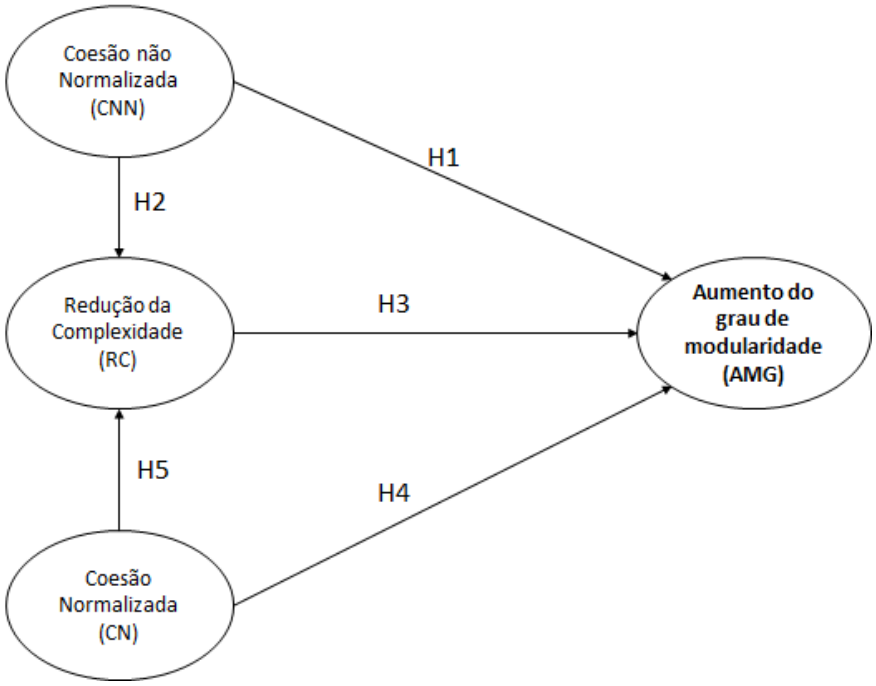


Figura 20 Manutenibilidade de software com aumento do grau de modularidade baseada nas características de qualidade interna de software

9.2.1 Coesão não normalizada (CNN)

As medidas de coesão não normalizada (**CNN**) são as medidas que consideram a coesão, baseada na contagem da quantidade distinta de atributos acessados, utilizando os métodos de classes em análise (AL DALLAH, 2013; GUPTA; CHHABRA, 2012). Por não serem normalizadas, seus valores não estão compreendidos por limites inferiores e superiores ([0..∞]) (GUPTA; CHHABRA, 2012). Chidamber e Kemerer definem a coesão de uma classe como o grau de similaridade entre os métodos (CHIDAMBER; KEMERER, 1991; CHIDAMBER; KEMERER, 1994). Além disso, apresentaram uma medida para coesão, denominada **LCOM**. Alguns pesquisadores propuseram melhorias em medidas estabelecidas (CHIDAMBER; KEMERER, 1994;

HENDERSON-SELLERS, 1996; HITZ; MONTAZERI, 1995; LI; HENRY, 1993). Dentre as suas variações mais conhecidas, está a medida **LCOM4** (HITZ; MONTAZERI, 1995).

As medidas **LCOM** e **LCOM4** correspondem à quantidade de pares de métodos de classes que não utilizam atributos em comum. Essas medidas visam detectar classes problemáticas. Um valor alto de **LCOM/LCOM4** indica baixa coesão e um valor baixo de **LCOM/LCOM4** indica alta coesão. Além disso, a medida **LCOM4** é uma variação da medida **LCOM**, que representa invocações de métodos que utilizam o mesmo atributo. Além disso, valores baixos para medidas de coesão não normalizadas podem estar relacionados à complexidade e ao baixo grau de modularidade do software (AL DALLAH, 2013; BRIAND et al., 2000). Essas medidas foram utilizadas em estudos para identificar aspectos de qualidade como propensão a falhas (AL DALLAH, 2011a; AL DALLAH, 2011b; AL DALLAH, 2012a; AL DALLAH, 2013; BRIAND et al., 2000). Dessa forma, propõe-se:

Hipótese 1: *As medidas LCOM e LCOM4 são boas indicadoras de coesão em sistemas de software OO open-source e possuem efeito significativo no aumento do grau de modularidade.*

Hipótese 2: *As medidas LCOM e LCOM4 são boas indicadoras de coesão em sistemas de software OO open-source e possuem efeito significativo na redução da complexidade*.

9.2.2 Redução da complexidade (RC)

A complexidade em um sistema de software pode determinar o quão confiável e sustentável pode ser o código (HONGLEI; WEI; YANAN, 2009).

Uma ampla gama de medidas de software na literatura é utilizada para verificar a complexidade do código de software em diferentes partes do sistema (KEVREKIDIS et al., 2009). Há estudos que indicam que medidas de tamanho e de complexidade são correlacionadas entre si (JALOTE, 2005; LEE; YANG, 2007). Por exemplo, a complexidade ciclomática corresponde à complexidade relativa de um método em relação a outro (MCCABE, 1976). Dessa forma, um método com uma quantidade maior de linhas de código pode ser mais complexo. No entanto, é possível afirmar que um método com complexidade ciclomática menor, normalmente, é menos complexo.

Além disso, há estudos que indicam correlação entre medidas de acoplamento, de complexidade e de tamanho (BASILI; BRIAND; MELO, 1995; GOEL; BHATIA, 2013; OFFUTT; ABDURAZIK; SCHACH, 2008; PANIGRAHI; BABOO; PADHY, 2015). Por exemplo, a medida **RFC** corresponde à quantidade de métodos distintos e de construtores invocados por uma classe. Então, um software que possui valores da medida **RFC** e a quantidade de métodos, de atributos e de classes (em termos de linhas de código) grande, a quantidade de métodos invocados tende a ser grande, aumentando a complexidade do código. Dessa forma, propõe-se:

Hipótese 3: *As medidas de complexidade, de tamanho e a medida de acoplamento RFC são boas indicadoras de complexidade em sistemas de software OO open-source e possuem efeito significativo no aumento do grau de modularidade*.

9.2.3 Coesão normalizada (CN)

As medidas de coesão normalizada (**CN**) são as medidas que consideram um aspecto de coesão e uma abordagem de medição diferente das medidas de

coesão não normalizada (**CNN**). **TCC** e **LCC** são duas medidas de coesão para classes, com base no conceito da semelhança entre cada par de métodos, em termos da razão entre os atributos comuns compartilhados (BIEMAN; KANG, 1995). Essas medidas são de coesão normalizadas (possuem limite superior e limite inferior [0..1]) e pretendem indicar a diferença entre "boa coesão" e "má coesão", em que valores próximos de 1 indicam alta coesão e valores próximos de 0 indicam baixa coesão.

O cálculo dessas medidas é mais restrito, pois não inclui construtores e destrutores, além de ser menos influenciado pelo tamanho. Além disso, a medida **LCC** considera pares de métodos indiretamente conectados. Os valores baixos para medidas de coesão normalizadas, assim como para as medidas de coesão não normalizadas, podem estar relacionadas à complexidade e ao baixo grau de modularidade do software (AL DALLAH, 2013; BRIAND et al., 2000). Há relatos na literatura que essas medidas foram utilizadas em estudos sobre reusabilidade e para a detecção de classes com propensão a falhas (AL DALLAH, 2011a; AL DALLAH, 2011b; AL DALLAH, 2012a; AL DALLAH, 2013; BRIAND et al., 2000; GUPTA; CHHABRA, 2012). Dessa forma, propõe-se:

Hipótese 4: *As medidas TCC e LCC são boas indicadoras de coesão em sistemas de software OO open-source e possuem efeito significativo no aumento do grau de modularidade*.

Hipótese 5: *As medidas TCC e LCC são boas indicadoras de coesão em sistemas de software OO open-source e possuem efeito significativo na redução da complexidade*.

9.2.4 Aumento do grau de modularidade (AGM)

Nas subseções anteriores, foram descritos fatores que podem contribuir para que o grau de modularidade do software aumente, alta coesão (não normalizada e normalizada) e baixa complexidade. Entretanto, como esses fatores podem descrever essa característica?

A modularização é um mecanismo para melhorar a flexibilidade e a compreensibilidade de um sistema, permitindo que o tempo de desenvolvimento e de manutenção possa ser reduzido (PARNAS, 1972). Há estudos que indicam que a arquitetura modular é um dos fatores de sucesso de sistemas de software OS de alta qualidade (DEKOENIGSBERG, 2008; GURBANI; GARVERT; HERBSLEB, 2005; LAWRIE; GACEK, 2002; PAECH; REUSCHENBACH, 2006). Para que um sistema de software tenha arquitetura madura, sustentável e reutilizável, quatro atributos de qualidade estão diretamente relacionados (EMANUEL et al., 2011): acoplamento/dependência, complexidade, tamanho e coesão.

Além dos fatores descritos, o desenvolvedor deve considerar a relação entre o acoplamento e a coesão: baixo acoplamento e alta coesão são importantes para sistemas modularizados (AL DALLAH, 2013). O aumento do grau de modularidade do sistema tem como indicadores medidas desses atributos de qualidade. As medidas **FOUT** e **LCOM2** tendem a ser indicadores dessas características, como identificado neste estudo e em estudos anteriores (AL DALLAH, 2011a; AL DALLAH, 2011b; AL DALLAH, 2012a; AL DALLAH, 2013; BRIAND et al., 2000; GUPTA; CHHABRA, 2012).

9.3 Metodologia do experimento

Nesta subseção, a descrição das variáveis de medição e dos dados utilizados para mensurar o modelo construído são apresentados.

9.3.1 Medição

O *dataset* do estudo foi constituído por sistemas de software *open-source* desenvolvidos em Java, selecionados a partir de Github e Sourceforge, dois repositórios mais populares da web. A coleta de dados foi realizada no período de 01/10/13 a 31/12/14. O Sourceforge é um repositório para o desenvolvimento e distribuição de software de código aberto, que possui mais de 430 mil projetos e hospeda mais de 3.7 milhões de usuários registrados (SOURCEFORGE, 2015). O Github é um serviço de hospedagem compartilhado para sistemas de software de código aberto, que utiliza o controle de versionamento Git. Oferece dois planos pagos (*Personal* e *Organizational*) para repositórios privados e contas gratuitas. Em 2014, o Github relatou ter mais de 3.4 milhões de usuários e com 16.7 milhões de projetos (GITHUB, 2015).

As medidas utilizadas no estudo foram identificadas em uma Revisão Sistemática de Literatura (RSL), cujo interesse de pesquisa foi identificar estudos sobre avaliação da qualidade interna de sistemas de software OO, utilizando técnicas/modelos estatísticos. Como resultado da RSL, foi obtido o total de 8.231 trabalhos, dos quais 79 trabalhos (estudos) foram selecionados como estudos primários. Na fase Análise dos Resultados, foram identificadas 265 medidas utilizadas para quantificar atributos de qualidade em sistemas de software. Entretanto, 15 medidas (as mais citadas) foram consideradas como as mais utilizadas em estudos para a avaliação da qualidade interna de software. Os fatores de qualidade interna de software foram encontrados em uma análise

fatorial executada na Seção 8, que utilizou como variáveis, as medidas identificadas nessa RSL.

9.3.2 Dados

Foi coletado um total de 1.031 sistemas de software categorizados em: *Audio & Video* (105 sistemas), *Business & Enterprise* (124 sistemas), *Communications* (112 sistemas), *Development* (132 sistemas), *Home & Education* (80 sistemas), *Games* (112 sistemas), *Graphics* (89 sistemas), *Science & Engineering* (93 sistemas), *Security & Utilities* (83 sistemas), *System Administration* (101 sistemas). Para cada uma das 10 categorias ou domínios de software propostos pelo Sourceforge, foram selecionados aleatoriamente sistemas de software para serem medidos, seguindo o fator de proporcionalidade populacional:

$$n_i = (N_i/N) \times n$$

sendo n_i o tamanho da amostra no estrato i, N_i o tamanho do estrato populacional i, N o tamanho total da população em estudo e n o tamanho total da amostra coletada para cada domínio.

Dessa forma, a amostra utilizada no estudo é composta por 500 sistemas de software em que 51 sistemas de *Audio & Video*, 60 sistemas de *Business & Enterprise*, 54 sistemas de *Communications*, 64 sistemas de *Development*, 39 sistemas de *Home & Education*, 54 sistemas de *Games*, 43 sistemas de *Graphics*, 45 sistemas de *Science & Engineering*, 40 sistemas de *Security & Utilities* e 49 sistemas de *System Administration*. Essa amostra possui em torno de 229.170 classes e totalizam mais de 20.838.192 de linhas de código. Os sistemas de software do Github foram organizados nos domínios com base na

descrição em seu repositório, uma vez que o repositório não categoriza os sistemas por domínios, mas por linguagem de programação.

Após a execução da análise fatorial, foi verificada que apenas 12 das 15 medidas identificadas como as mais citadas em estudos que investigam a qualidade interna de sistemas de software possuem correlação entre si. Logo, a descrição dos fatores de qualidade interna de software e as medidas utilizadas como indicadoras desses fatores são determinantes para a caracterização do modelo apresentado. Na Tabela 26 e na Tabela 27, a operacionalização dos construtos é resumida, utilizando itens adaptados da RSL.

Tabela 26 Operacionalização dos construtos do modelo – conjunto I

Construtos	Item	Descrição	Referências
Coesão Normalizada	*CN1*	Calcula a quantidade de ligações diretas (quantidade de métodos conectados entre si), ou seja, a densidade das ligações (TCC).	(Bieman; Kang, 1995)
	CN2	Calcula a quantidade de métodos ligados de forma direta ou indireta em uma classe (LCC)	(Bieman; Kang, 1995)
Aumento do Grau de Modularidade	*AGM1*	Calcula a quantidade de classes referenciadas por uma classe em um pacote (FOUT).	(Mubarak et al., 2010)
	AGM2	Calcula utilizando o método de Henderson-Sellers em que se calcula a média da quantidade de métodos que acessam um atributo, subtraindo a quantidade de métodos e dividindo o resultado por (1-M) (LCOM2).	(Henderson-Sellers, 1996)
Coesão não Normalizada	*CNN1*	Para cada par de métodos na classe, verifica o acesso a conjuntos disjuntos de variáveis de instância (LCOM).	(Chidamber; Kemerer, 1991; Chidamber; Kemerer, 1994)
	CNN2	Calcula a variação da medida LCOM, LCOM4. Mede a quantidade de "componentes ligados" em uma classe.	(Hitz; Montazeri, 1995)

Tabela 27 Operacionalização dos construtos do modelo – conjunto II

Construtos	Item	Descrição	Referências
Redução da Complexidade	*RD1*	Calcula a quantidade total de linhas de código de um projeto. Não são contabilizados espaços em branco e comentários (LOC).	(Briand et al., 2000; Al Dallah, 2013)
	RD2	Calcula a quantidade total de métodos em um projeto (NOM).	(Briand et al., 2000; Al Dallah, 2013)
	RD3	Calculaa quantidade total de atributos em um projeto (NOA).	(Briand et al., 2000; Al Dallah, 2013)
	RD4	Conta a quantidade de fluxos em um bloco de código (CC).	(McCabe, 1976)
	RD5	Calcula a soma a complexidade ciclomática dos métodos em uma classe (WMC).	(Chidamber; Kemerer, 1991; Chidamber; Kemerer, 1994)
	RD6	Calcula a quantidade de métodos e construtores distintos invocados por uma classe (RFC).	(Chidamber; Kemerer, 1991; Chidamber; Kemerer, 1994)

9.4 Resultados

Para testar as hipóteses decorrentes do modelo conceitual, um modelo de equações estruturais (SEM) foi utilizado. Uma vez que o *dataset* não foi distribuído normalmente ($p < 0,01$ com base no teste de *Kolmogorov-Smirnov*), o modelo de pesquisa foi desenvolvido anteriormente e não tinha sido previamente testado, o método PLS foi considerado adequado (CHIN, 1998; CHIN; MARCOLIN; NEWSTED, 2003). Além disso, a aplicação do PLS requer um tamanho de amostra igual ou maior que: i) 10 vezes a quantidade de itens contidos nos construtos; ou ii) 10 vezes que a quantidade de caminhos estruturais dirigidos a um particular construto no modelo (CHIN, 1998; GEFEN; STRAUB, 2005).

No modelo conceitual elaborado, a maior quantidade de caminhos estruturais dirigidos a um particular construto é três (construto **Aumento do Grau de Modularidade**), o que significa que a amostra mínima requerida é de 30 casos. Dado que a população da amostra foi maior (n = 500), a utilização do método PLS foi considerada adequada. Antes de testar o modelo conceitual, o modelo de medição foi analisado, utilizando o software *Smart-PLS 2.0*, para avaliar a consistência interna, o indicador de confiabilidade, a validade convergente e a validade discriminante.

9.4.1 Modelo de medição

Inicialmente, a consistência interna do modelo foi avaliada por meio do teste da confiabilidade composta (CR), cujo valor deve ser maior do que 0,70 (HAIR et al., 2014; WERTS; LINN; JORESKOG, 1974). Conforme apresentado na Tabela 28, o CR mínimo foi **0,8779**, indicando que o modelo possui consistência interna (STRAUB, 1989). O segundo passo foi a avaliação do indicador de confiabilidade, baseado no critério de que as cargas dos indicadores devem ser superiores a 0,70. Caso houver cargas menores que 0,4, elas devem ser eliminadas (CHURCHILL, 1979; HENSELER; RINGLE; SINKOVICS; 2009). No modelo proposto, os indicadores possuíam cargas com valores acima de 0,70. Na Tabela 28, as cargas em negrito superiores ou marginais ao valor 0,70 são apresentadas, confirmando a confiabilidade dos indicadores (as medidas de software) do modelo.

Tabela 28 Índices do PLS e *Cross-Loadings* gerados

Construtos	Medidas	RC	AGM	CNN	CN
Redução da Complexidade (RC) **CR=0,9245; AVE=0,6729**	*LOC*	**0,8722**	0,132	0,081	0,0167
	NOM	**0,702**	0,0653	0,0318	0,1325
	NOA	**0,9197**	0,2941	0,183	-0,0789
	CC	**0,8185**	0,0748	0,0514	0,0485
	WMC	**0,7563**	0,2556	0,0123	0,0766
	RFC	**0,8342**	0,1506	0,1419	-0,1032
Aumento do Grau de Modularidade (AGM) **CR=0,8779; AVE=0,7824**	*FOUT*	0,2473	**0,8753**	0,0569	0,1647
	LCOM2	0,1859	**0,8937**	0,186	0,1983
Coesão não Normalizada (CNN) **CR=0,9903, AVE=0,9807**	*LCOM*	0,0782	0,1418	**0,9877**	-0,0337
	LCOM4	0,159	0,1362	**0,9929**	-0,0091
Coesão Normalizada (CN) **CR=0,9711, AVE=0,9439**	*TCC*	-0,0172	0,2042	-0,015	**0,9728**
	LCC	-0,0159	0,1955	-0,0239	**0,9703**

Em seguida, a validade convergente foi analisada, o que requer que a variância média extraída (*Average Variance Extracted* - AVE) seja superior a 0,5 (Tabela 28) (HAIR et al., 2014; KWONG; WONG, 2013). Por fim, a validade discriminante foi avaliada, com base em dois critérios (FORNELL; LARCKER, 1981): i) as cargas de cada indicador (em negrito) devem ser maior que o *cross-loading*; e ii) a raiz quadrada da AVE, para cada construto, deve ser maior do que as correlações com os construtos. Por exemplo, na Tabela 28, o valor da AVE para o construto **Aumento do Grau de Modularidade** é **0,7824**. Dessa forma, a raiz quadrada do construto é **0,885** (Tabela 29). Esse resultado é maior que os valores de correlação apresentados na coluna **AGM** (**0,6435**; **0,1400**; **0,5058**) e é maior do que o valor da linha **AGM** (omitido, pois é zero, ou seja, não existe caminho estrutural entre **Aumento do Grau de Modularidade** e outros construtos). Dessa forma, conclui-se que os construtos mostram evidências de discriminação aceitável.

Para a avaliação formativa dos modelos de medição, foram avaliadas a multicolinearidade (as variáveis independentes possuem relações lineares exatas ou aproximadamente exatas), a significância e os pesos. Em relação à multicolinearidade, o fator de variância inflacionada (*Variance Inflation Factor* - VIF) está abaixo do valor de corte de 5,0 (HAIR; RINGLE; SARSTEDT, 2011). Em relação à significância e aos pesos, as 10 medidas que compõem os construtos como indicadores possuem pesos entre **0,221** e **0,743**, indicando que os indicadores formativos foram estatisticamente significativos. Por exemplo, para as variáveis dependentes **FOUT** e **LCOM2**, os valores estão abaixo do valor de corte sugerido. Isso significa que coeficientes da regressão são estatisticamente significativos, segundo a estatística *p* convencional. Esses valores são apresentados na Tabela 30.

Tabela 29 Correlações e raízes quadradas da AVE (em negrito)

Construtos	**RC**	**AGM**	**CNN**	**CN**
Redução da Complexidade (RC)	**0,820**	0,6435	0,1252	-0,0171
Aumento do Grau de Modularidade (AGM)	-	**0,885**	-	-
Coesão não Normalizada (CNN)	-	0,1400	**0,990**	-0,0199
Coesão Normalizada (CN)	-	0,5058	-	**0,972**

Tabela 30 Indices de tolerância e valores do coeficiente VIF

Variáveis Dependentes (Construto - Aumento do Grau de Modularidade)	**Variáveis Independentes (Indicadores)**	**Tolerância**	**VIF**
FOUT LCOM2	LOC	0,625	1,600
	NOM	0,426	2,350
	NOA	0,221	4,522
	CC	0,743	1,346
	WMC	0,498	2,008
	RFC	0,276	3,628
	LCOM	0,243	4,108
	LCOM4	0,653	1,531
	TCC	0,245	4,076
	LCC	0,669	1,495

Em resumo, os testes mostraram que o modelo tem boa consistência interna, bom indicador de confiabilidade, validade convergente e discriminante. Além disso, os construtos possuem boas propriedades. Consequentemente, os construtos desenvolvidos no modelo de medição podem ser utilizados para testar o modelo estrutural.

9.4.2 Modelo estrutural

Com base na validação do modelo de medição, a significância dos caminhos do modelo estrutural foi testada utilizando uma inicialização com 5.000 subamostras e o procedimento *bootstrapping*. Para um teste com nível de significância de 1%, o coeficiente de caminho estrutural será significativo se o valor *p* for maior do que **1,96** (HAIR et al., 2014; KWONG; WONG, 2013). Com o resultado do procedimento *bootstrapping* pode ser observado que o relacionamento **Coesão Normalizadas (CN) → Redução da Complexidade (RC)** não é significativo (***p* = 0,267**). Na Figura 21, os coeficientes dos caminhos

(valores padronizados) e os valores *p* (entre parênteses) para as hipóteses propostas são mostrados.

O critério utilizado para avaliar a capacidade de previsão do modelo estrutural (HAIR et al., 2014; KWONG; WONG, 2013) requer resultado acima de 0,2. Conforme pode ser visualizado na Figura 21, os construtos dependentes estão acima do índice requerido. O modelo explica 11,6% do aumento do grau de modularidade e duas das cinco hipóteses foram confirmadas (ou seja, H3 e H4) e as hipóteses H1, H2 e H5 não foram confirmadas.

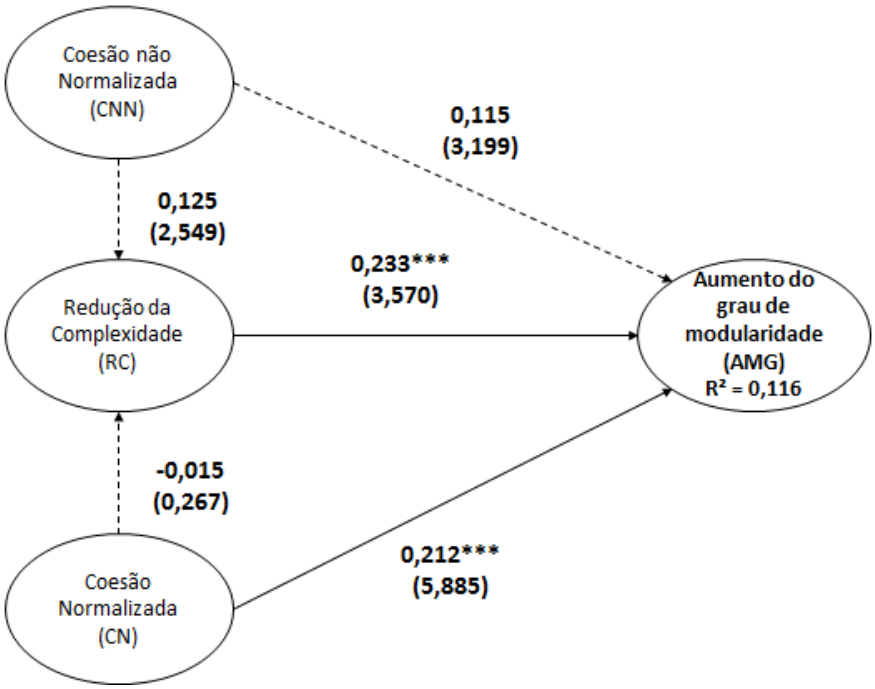


Figura 21 Modelo conceitual com os coeficientes dos caminhos (relacionamento entre os construtos)

***p < 0.01. Para evitar um grafo populoso, os indicadores para cada construto não são mostrados.

Em relação à eficácia dos construtos independentes, para explicar o aumento do grau de modularidade, nota-se que a **Redução da Complexidade (RC)** tende a ser mais relevante (***β* = 0,233**), seguido da **Coesão Normalizada (CN)** (***β* = 0,212**). No geral, das cinco hipóteses formuladas, duas são confirmadas pelos dados. O resumo das hipóteses pode ser visualizado na Tabela 31.

Tabela 31 Hipóteses sumarizadas

Hipóteses	Descrição	Confirmada
H1	As medidas LCOM e LCOM4 são boas indicadoras de coesão em sistemas de software OO *open-source* e possuem efeito significativo no aumento do grau de modularidade.	Não
H2	As medidas LCOM e LCOM4 são boas indicadoras de coesão em sistemas de software OO *open-source* e possuem efeito significativo na redução da complexidade.	Não
H3	As medidas de complexidade, tamanho e a medida de acoplamento RFC são boas indicadoras de complexidade em sistemas de software OO *open-source* e possuem efeito significativo no aumento do grau de modularidade.	Sim
H4	As medidas TCC e LCC são boas indicadoras de coesão em sistemas de software OO *open-source* e possuem efeito significativo no aumento do grau de modularidade.	Sim
H5	As medidas TCC e LCC são boas indicadoras de coesão em sistemas de software OO *open-source* e possuem efeito significativo na redução da complexidade.	Não

9.5 Discussão e conclusões

Os resultados da validação do modelo confirmam a importância de determinados fatores para o **Aumento do Grau de Modularidade (AGM)**: **Redução da Complexidade (RC)** e **Coesão Normalizada (CN)**. Dessa forma, os construtos **RC** (H3) e **CN** (H4) explicam o construto **AGM** em 11,6%. Entretanto, não foi possível determinar se a **Coesão não Normalizada (CNN)** é

significativa para **AGM** (H1) e para **RC** (H2). Também não foi possível determinar se a **CN** é significativa para **RC** (H5). Em outras palavras, os resultados do modelo não confirmaram se **CNN** e **CN** podem explicar as variâncias de **AGM** e **RC**, respectivamente.

Assim, quanto menor a complexidade e maior a coesão, maior o grau de modularidade do sistema de software, a sua manutenibilidade, a sua capacidade de evoluir e ser reusável. O resultado encontrado no modelo estrutural (Figura 21) indica que as revisões no código, buscando rever relações diretas e indiretas de métodos que utilizam o mesmo atributo, são mais efetivas para aumentar a coesão de uma classe. Isso significa que as medidas **TCC** e **LCC** são mais relevantes para indicar ausência ou não de coesão em uma classe. Além disso, quando os sistemas são de alta complexidade (valores altos para as medidas **CC**, **WMC**, **LOC**, **NOA** e **NOM**) e têm problemas de modularidade (valores altos para **RFC**), significa que a estrutura da classe é extensa quanto às suas ligações e o tamanho e que há muitas dependências entre classes e pacotes. Essa estrutura pode ser refatorada, por exemplo, reduzindo a quantidade de atributos globais, de linhas de código e de blocos de métodos. Deve ser feita também separação de responsabilidades nas classes (não acumular funções em classes), o que contribui para que o sistema de software possa ser modular.

No modelo estrutural, a **Coesão não Normalizada** não é relevante para explicar o **Aumento do Grau de Modularidade**. O resultado, portanto, está alinhado com descobertas anteriores na literatura, sobre a capacidade das medidas de qualidade interna **LCOM** e **LCOM4** na avaliação e na predição da modularidade e da complexidade de sistemas de software. Uma possível explicação para esses resultados pode ser encontrada na literatura (AL DALLAL, 2010; AL DALLAL; BRIAND, 2010; BRIAND; DALY; WUEST, 1998). A medida **LCOM** é relativamente fraca na predição de características de qualidade externa, pois não detecta diferenças entre os diferentes tipos de

métodos (métodos especiais e de acesso, como os *getters* e *setters*). A medida **LCOM4** é baseada na contagem da quantidade de interações entre os métodos e não faz distinção entre os dois componentes relacionados que têm a mesma quantidade de nós (métodos) e arestas (relacionamento entre métodos que compartilham pelo menos um atributo). Consequentemente, tende a ter fraco poder discriminativo.

Embora as medidas analisadas sejam as mais utilizadas em estudos empíricos, sobre a qualidade interna de software, é possível que outras medidas com as mesmas propriedades (por exemplo, acoplamento, coesão etc.) ou derivações e adaptações similares possam ter melhores resultados. Por exemplo, a medida **LCOM** possui derivações (**LCOM2**, **LCOM3**, **LCOM4**, **LCOM5**, **ILCOM** etc.), sendo que duas delas foram utilizadas no modelo proposto, por estarem entre as medidas mais utilizadas por pesquisadores, para a avaliação da qualidade interna de software.

Além da medida **LCOM**, outras medidas podem ter variantes ou derivações sendo indicadores mais significativos da qualidade interna de software, ainda que não sejam amplamente utilizadas por pesquisadores. Por exemplo, a medida **CBO** não apresentou correlação com as medidas indicadoras nos construtos do modelo encontrados nesse estudo. Porém, os resultados da RSL mostraram que há variações da medida como **CBO'**, **CBO_IUB** e **CBO_U** (AL DALLAH, 2013; BRIAND et al., 2000), sendo utilizadas em estudos de qualidade de software.

9.5.1 Implicações práticas

As conclusões do estudo sugerem um conjunto de fatores que formam um modelo de equações estruturais para explicar a manutenibilidade de sistemas de software OO. Na prática, esse modelo multivariado construído pode ser

integrado em ambientes de desenvolvimento de software (IDE's), para estimar a manutenibilidade de classes sob três fatores: i) redução da complexidade; ii) coesão normalizada; e iii) aumento do grau da modularidade. Dessa forma, os resultados encontrados no modelo multivariado podem ser implantados no *plug-in* desenvolvido para extrair os valores das medidas para indicar a probabilidade de uma classe necessitar de revisões.

Para os engenheiros de software, os resultados do modelo podem auxiliar o processo de remodelagem do domínio de negócio para que sejam levemente acoplados e altamente coesos. Para os arquitetos de software, o modelo pode auxiliar na localização de classes com baixa manutenibilidade e direcionar os testes para reduzir as chances de detecção de problemas, durante a fase de manutenção. Além disso, os gerentes de projeto podem utilizar e incorporar esses indicadores de qualidade como fatores a serem considerados no processo de gestão de qualidade dos projetos da organização. Por fim, os indicadores de qualidade podem ser utilizados como referências para classes com baixa manutenibilidade e que exigem refatoração. Os desenvolvedores de software devem ser encorajados a documentar essas classes para executar as revisões necessárias.

Nesses cenários práticos descritos, os engenheiros, os arquitetos, os gestores e os desenvolvedores, necessitam de medidas que possuem correlação e significância estatística para serem capazes de distinguir a qualidade entre diferentes classes ou nas implementações da mesma classe. Caso contrário, quando essa diferença não é notada, a tomada de decisões sobre a refatoração de classes no sistema de software pode ser incorreta ou as ações executadas na codificação podem enfraquecer a qualidade do código desenvolvido.

9.5.2 Implicações teóricas

Os resultados do PLS-SEM mostram a possibilidade de construir modelos multivariados estatisticamente significativos com boa consistência interna, bom indicador de confiabilidade, validade convergente e discriminante e construtos com boas propriedades, obtidos previamente em uma Análise Fatorial Exploratória (AFE). Os modelos multivariados tendem a ser mais gerais, pois um único atributo de qualidade interna de software pode não ser capaz de caracterizar os aspectos de um atributo de qualidade externa, por exemplo, a manutenibilidade.

A partir dessa perspectiva, acredita-se que o modelo proposto fornece informações úteis em relação a quatro atributos de qualidade interna, isto é, o tamanho, a complexidade, o acoplamento e a coesão. Geralmente, os modelos multivariados são gerais na sua aplicabilidade. Essa expectativa é confirmada pelos resultados da sua validação, que mostraram que a maioria dos fatores propostos são significantes para a caracterização da qualidade de projetos em diferentes domínios de aplicação. Assim, pode-se concluir que:

a) Os construtos **Coesão Normalizada (CN)** e **Redução da Complexidade (RC)** são preditores do construto **Aumento do Grau de Modularidade (AGM)**. As medidas **TCC**, **LCC**, **CC**, **WMC**, **LOC**, **NOA**, **NOM** e **RFC** são boas indicadoras dessas características;
b) O construto **Coesão não Normalizada (CNN)** não prediz os construtos **AGM** e **RC**;
c) O construto **CN** não prediz o construto **RC** diretamente.

9.5.3 Ameaças à validade

Apesar dos resultados do estudo contribuirem para diversas áreas do desenvolvimento de software e no suporte a garantia da manutenibilidade de sistemas de software, o estudo realizado possui algumas limitações ou ameaças à validade. Podem ser identificadas algumas ameaças à validade, por exemplo:

a) Quanto à validade do construto:

- O modelo proposto não engloba todas as propriedades de um sistema de software OO, por exemplo, a herança. Embora as medidas dessa propriedade tenham sido identificadas como as mais citadas e utilizadas em estudos empíricos sobre a qualidade interna de software, ao executar a análise fatorial (base para os construtos do modelo), não foi possível identificar correlações entre elas e outras medidas estudadas;
- Para o PLS-SEM, embora suportem a maioria das relações entre os construtos, é possível que o modelo proposto não possua impacto substancial na qualidade de sistemas de software OO OS, uma vez que um estudo qualitativo do modelo não estava no escopo do trabalho.

b) Quanto à validade interna:

O resultado mostra evidências empíricas, significado prático das similaridades e características dos sistemas de software OO OS, mas não se aprofunda em detalhes técnicos, que tais resultados possam ser comprovados com uma análise qualitativa nos sistemas analisados;

c) Quanto à validade externa:

O estudo tem como premissa analisar sistemas de software OO, mas apenas considera sistemas desenvolvidos na linguagem Java.

9.6 Considerações finais

Nesta seção, um modelo de equações estruturais foi proposto utilizando construtos identificados em uma AFE e validado pelo PLS-SEM. Esse modelo propõe uma relação de medidas que pode fornecer informações úteis aos desenvolvedores em relação a quatro atributos de qualidade interna: o tamanho, a complexidade, o acoplamento e a coesão. A partir dos resultados do modelo, a correlação dessas medidas pode ser avaliada na tomada de decisões para aprimorar a manutenibilidade de um sistema de software OO, melhorando a qualidade interna desses sistemas.

Embora os resultados de PLS-SEM suportem a maioria dos fatores extraídos na AFE, uma investigação mais aprofundada deve ser considerada para explorar qualitativamente e validar se tais recomendações possuem impacto substancial na qualidade de sistemas. Além disso, o estudo utilizou apenas sistemas de software OO desenvolvidos em Java. Essa observação implica que as medidas consideradas podem ser relevantes apenas para esses sistemas.

10 CONSIDERAÇÕES FINAIS

10.1 Conclusões

A garantia de qualidade de sistemas de software OO tem sido considerada um fator crítico para empresas desenvolvedoras de software. As atividades relacionadas a essa garantia vêm sendo tratadas como prioridade no processo de desenvolvimento, pois a qualidade não é mais vista como um fator de vantagem competitiva. Ela tornou-se uma condição necessária, se uma empresa deseja apresentar um sistema de alta qualidade, a baixos custos e dentro dos prazos estipulados para os seus usuários. De forma específica, a qualidade interna do código (o quão bem escrito ele é) influencia a qualidade do sistema de software. Apesar de não ser diretamente percebida pelo usuário final, a qualidade desses sistemas pode ser construída de forma indireta, por exemplo, no tempo de manutenção ou de evolução do software promovido pela equipe de desenvolvimento.

Neste trabalho foi proposto um Modelo de Equações Estruturais (*Structural Equation Modeling* - SEM) capaz de refletir as relações estruturais entre as medidas de software analisadas, cuja relevância e reconhecimento de qualidade por pesquisadores da área foram estudados e validados por uma Revisão Sistemática de Literatura (RSL). Os valores para essas medidas foram coletados pelo *plug-in O3SMeasures*, a partir da avaliação de sistemas de software OO Java, considerando diferentes domínios. Das 16 medidas implementadas, uma delas (*Number of Classes* - NC) foi utilizada para fins descritivos da população estudada.

Em relação aos objetivos específicos:

a) “Identificar quais medidas de software aplicadas à tecnologia OO são relevantes para a caracterização da qualidade interna de sistemas de software OS quanto à característica Manutenibilidade”, os resultados da RSL sugerem que há 15 medidas associadas a cinco propriedades diferentes que podem ser consideradas como as mais utilizadas por pesquisadores da área para quantificar a qualidade de software, por exemplo, as medidas **RFC**, **LCOM2**, **TCC** e **LCC**. Essas medidas foram identificadas utilizando o critério de citação em 10 ou mais trabalhos científicos na última década;

b) “Identificar fatores que caracterizam a qualidade interna de sistemas de software OO OS, quanto à sua manutenibilidade, com base nos resultados obtidos nas etapas exploratórias e na avaliação do estudo”, a solução extraída da Análise Fatorial indica que há quatro fatores que caracterizam um sistema de software manutenível: i) **Redução da Complexidade**; ii) **Coesão não Normalizada**; iii) **Coesão Normalizada**; e iv) **Aumento do Grau de Modularidade**. Esses fatores explicam a correlação entre 12 das 15 medidas de software estudadas, indicando que existe tendência comportamental do código fonte desenvolvido nesses sistemas, quanto à coesão, à modularidade e à complexidade desses sistemas;

c) “Identificar o relacionamento entre fatores em termos teóricos e verificar a correspondência estatística dos dados coletados em sistemas de software OO OS”, o modelo criado utilizando a técnica PLS-SEM indica que os fatores **Redução da Complexidade** e **Coesão Normalizada** são significativos para o **Aumento do Grau de Modularidade**. Isso significa que as medidas que compõem os construtos são bons indicadores da ausência ou não de características de qualidade. Entretanto, não foi possível determinar se **Coesão não**

Normalizada é significativa para **Aumento do Grau de Modularidade** e para **Redução da Complexidade**. Também não foi possível determinar se **Coesão Normalizada** é significativa para **Redução da Complexidade**. Dessa forma, esses construtos podem ser considerados como recomendações para a adoção de práticas e garantir a manutenibilidade e a qualidade interna de sistemas de software OO OS desenvolvidos.

10.2 Contribuições

A contribuição deste estudo para a área Engenharia de Software se dá pela complementação, por meio de técnicas quantitativas, de observações pautadas na literatura para aspectos estruturais da tecnologia de OO, como a modularidade, a abstração e a complexidade. De forma sucinta, as principais contribuições foram:

a) Uma **RSL** sobre as principais medidas e técnicas estatísticas utilizadas para quantificar propriedades e caracterizar a qualidade interna de sistemas de software OO OS;
b) Um ***plug-in*** para o **Eclipse** para medição de software OO com as 15 medidas mais relevantes para a caracterização da qualidade interna de sistemas;
c) A identificação de **fatores** por meio da **Análise Fatorial**, que podem caracterizar a qualidade de sistemas de software OO em relação à manutenibilidade;
d) Um **Modelo de Equações Estruturais**, desenvolvido e testado, utilizando a técnica **PLS** (**PLS-SEM**), validando empiricamente as correlações identificadas na Análise Fatorial.

10.3 Ameaças à validade

No estudo realizado, podem ser identificadas algumas ameaças à validade, por exemplo:

a) Quanto à validade do construto:

- Para a Análise Fatorial, mesmo que as medidas utilizadas tenham sua relevância justificada na literatura, por exemplo, as medidas DIT, NOC e CBO, não foram possíveis identificar correlações entre elas e entre outras medidas estudadas;
- Para o PLS-SEM, embora suportem a maioria das relações entre os construtos, é possível que o modelo proposto não possua impacto substancial na qualidade de sistemas de software OO OS, uma vez que um estudo qualitativo do modelo não estava no escopo do trabalho;

b) Quanto à validade interna:

- Em relação ao *plug-in O3SMeasures* desenvolvido, embora testado e validado internamente, existe a possibilidade de haver diferenças entre os resultados das medidas comparados a outros *plug-ins*;
- O resultado mostra evidências empíricas, significado prático das similaridades e das características dos sistemas de software OO OS, mas não se aprofunda em detalhes técnicos, que tais resultados possam ser comprovados com uma análise qualitativa nos sistemas analisados;

c) Quanto à validade externa:

O estudo tem como premissa analisar sistemas de software OO, mas apenas considera sistemas desenvolvidos na linguagem Java.

10.4 Trabalhos futuros

Como trabalhos futuros, sugere-se:

a) Utilização de outras medidas de software, a fim de obter novos resultados, que possam justificar características não exploradas no estudo;
b) Avaliar a qualidade interna sob outras características definidas na ISO/IEC 25010;
c) Desenvolver estudos qualitativos para validar as evidências encontradas no estudo, por exemplo, inspeções em código para validar as tendências de similaridade entre os domínios e um estudo de caso para validar a efetividade do modelo proposto.

10.5 Resultados de publicações

Os resultados preliminares do estudo foram publicados em dois *workshops* e em um simpósio:

a) SANTOS, M. A.; COSTA, H. A. X.; BERMEJO, P. H. S. **Internal Quality Factors of Object-Oriented Software Developed in Java**. In: Pre-ICIS sigIQ Workshop, 2013, Milan, Italy. Proceedings of Quality Information in Organizations and Society. p. 9-9, 2013;

b) SANTOS, M.; AMADOR, R.; BERMEJO, P. H. S.; COSTA, H. **Similar Characteristics of Internal Software Quality Attributes for Object-Oriented Open-Source Software Projects.** In: XIII Simpósio Brasileiro de Qualidade de Software, (SBQS 2014), Blumenau-SC. v. 13, pp. 209-223, 2014;

c) SANTOS, M.; BERMEJO, P.; COSTA, H. **Um Modelo Conceitual para a Caracterização da Qualidade Interna de Sistemas de Software Open-Source Orientado a Objetos.** In: Anais IV Workshop de Teses e Dissertações do CBSOFT - WTDSoft 2014, v. 2, pp. 68-73, 2014.

REFERÊNCIAS

ABUASAD, A.; ALSMADI, I. M. Evaluating the Correlation Between Software Defect and Design Coupling Metrics. In: INTERNATIONAL CONFERENCE ON COMPUTER, INFORMATION AND TELECOMMUNICATION SYSTEMS, 2012, Amman. **Proceedings…** Amman: IEEE, 2012. p. 1-5.

AGGARWAL, K. K. et al. Empirical study of object-oriented metrics. **Journal of Object Technology**, Oxford, v. 5, n. 8, p. 149-173, Nov./Dec. 2006.

AL DALLAL, J. Fault prediction and the discriminative powers of connectivity-based object-oriented class cohesion metrics. **Information and Software Technology**, London, v. 54, n. 4, p. 396-416, Apr. 2012a.

AL DALLAL, J. Improving the applicability of object-oriented class cohesion metrics. **Information and Software Technology**, London, v. 53, n. 9, p. 914-928, Sept. 2011a.

AL DALLAL, J. Mathematical validation of object-oriented class cohesion metrics. **International Journal of Computers**, Anaheim, v. 4, n. 2, p. 45-52, 2010.

AL DALLAL, J. Measuring the discriminative power of object-oriented class cohesion metrics. **IEEE Transactions on Software Engineering**, New York, v. 37, n. 6, p. 788-804, Nov./Dec. 2011b.

AL DALLAL, J. Object-oriented class maintainability prediction using internal quality attributes. **Information and Software Technology**, London, v. 55, n. 11, p. 2028-2048, Nov. 2013.

AL DALLAL, J. The impact of accounting for special methods in the measurement of object-oriented class cohesion on refactoring and fault prediction activities. **Journal of Systems and Software**, New York, v. 85, n. 5, p. 1042-1057, May 2012b.

AL DALLAL, J. Transitive-based object-oriented lack-of-cohesion metric. **Procedia Computer Science**, Amsterdam, v. 3, p. 1581-1587, Feb. 2011c.

AL DALLAL, J.; BRIAND, L. C. A precise method-method interaction-based cohesion metric for object-oriented classes. **ACM Transaction on Software Engineering an Methodology**, New York, v. 21, n. 2, p. 1-34, Mar. 2012.

AL DALLAL, J.; BRIAND, L. C. An object-oriented high-level design-based class cohesion metric. **Information and Software Technology**, London, v. 52, n. 12, p. 1346-1361, Dec. 2010.

ALLEN, E. B.; KHOSHGOFTAAR, T. M. Measuring coupling and cohesion: an information-theory approach. In: INTERNATIONAL SOFTWARE METRICS SYMPOSIUM, 6., 1999, Boca Raton. **Proceedings…** Boca Raton: [s.n.], 1999. p. 119-127.

AL-QUTAISH, R. E. Measuring the software product quality during the software development life-cycle: an international organization for standardization standards perspective. **Journal of Computer Science**, Lavras, v. 5, n. 5, p. 392-397, maio 2009.

AMPATZOGLOU, A.; CHATZIGEORGIOU, A. Evaluation of object-oriented design patterns in game development. **Information and Software Technology**, London, v. 49, n. 5, p. 445-454, May 2007.

ANDA, B. Assessing software system maintainability using structural measures and expert assessments. In: INTERNATIONAL CONFERENCE ON SOFTWARE MAINTENANCE, 2007, Paris. **Proceedings…** Paris: IEEE, 2007. p. 204-213.

APPOLINÁRIO, F. **Dicionário de metodologia científica:** um guia para a produção do conhecimento científico. São Paulo: Atlas, 2004. 320 p.

ARISHOLM, E.; BRIAND, L. C. Predicting fault-prone components in a java legacy system. In: INTERNATIONAL SYMPOSIUM ON EMPIRICAL SOFTWARE ENGINEERING, 2006, Rio de Janeiro. **Proceedings…** Rio de Janeiro: [s.n.], 2006. p. 8-17.

ATALLAH, N. A.; CASTRO, A. A. Revisões sistemáticas da literatura e metanálise: a melhor forma de evidência para tomada de decisão em saúde e a maneira mais rápida de atualização terapêutica. **Diagnóstico & Tratamento**, São Paulo, v. 2, n. 2, p. 12-15, 1997.

ATOLE, C. S.; KALE, K. V. Assessment of package cohesion and coupling principles for predicting the quality of object oriented design. In: INTERNATIONAL CONFERENCE ON DIGITAL INFORMATION MANAGEMENT, 1., 2006, Bangalore. **Proceedings…** Bangalore: [s.n.], 2006. p. 1-5.

BABICH, D. et al. Using a class abstraction technique to predict faults in OO classes: a case study through six releases of the eclipse JDT. In: SYMPOSIUM ON APPLIED COMPUTING, 2011, Taichung. **Proceedings…** Taichung: [s.n.], 2011. p. 1419-1424.

BADRI, L.; BADRI, M.; TOURÉ, F. An empirical analysis of lack of cohesion metrics for predicting testability of classes. **International Journal of Software Engineering and its Applications**, Tasmania, v. 5, n. 2, p. 69-85, Apr. 2011.

BADRI, M.; DROUIN, N.; TOURE, F. on understanding software quality evolution from a defect perspective: a case study on an open source software system. In: INTERNATIONAL CONFERENCE ON COMPUTER SYSTEMS AND INDUSTRIAL INFORMATICS, 2012, Sharjah. **Proceedings…** Sharjah: [s.n.], 2012. p. 1-6.

BAGGEN, R. et al. Standardized code quality benchmarking for improving software maintainability. **Software Quality Journal**, Essex, v. 20, n. 2, p. 287-307, June 2012.

BAGOZZI, R. P.; YI, Y. On the evaluation of structural equation models. **Journal of the Academy of Marketing Science**, Greenvale, v. 16, n. 1, p. 74-94, Mar. 1988.

BAILEY, J. et al. Evidence relating to object-oriented software design: a survey. In: INTERNATIONAL SYMPOSIUM ON EMPIRICAL SOFTWARE ENGINEERING AND MEASUREMENT, 2007, Madrid. **Proceedings…** Madrid: [s.n.], 2007. p. 482-484.

BAILEY, J. et al. Search engine overlaps: do they agree or disagree? In: INTERNATIONAL WORKSHOP ON REALISING EVIDENCE-BASED SOFTWARE ENGINEERING, 2007, Minneapolis. **Proceedings…** Minneapolis: [s.n.], 2007. p. 2.

BARKER, R.; TEMPERO, E. A Large-scale empirical comparison of object-oriented cohesion metrics. In: ASIA-PACIFIC SOFTWARE ENGINEERING CONFERENCE, 2007, Aichi. **Proceedings…** Aichi: [s.n.], 2007. p. 414-421.

BARKMANN, H.; LINCKE, R.; LOWE, W. Quantitative evaluation of software quality metrics in open-source projects. In: INTERNATIONAL CONFERENCE ON ADVANCED INFORMATION NETWORKING AND APPLICATIONS WORKSHOPS, 2010, Bradford. **Proceedongs…** Bradford: [s.n.], 2010. p. 1067-1072.

BASILI, V. Measuring the software process and product: lesson learned in the SEL. In: ANNUAL NASA SOFTWARE ENGINEERING WORKSHOP, 1985, Maryland. **Proceedings…** Maryland: [s.n.], 1985. p. 1-37.

BASILI, V. R.; BRIAND, L. C.; MELO, W. L. A validation of object-oriented design metrics as quality indicators. **IEEE Transactions on Software Engineering**, New York, v. 22, n. 10, p. 751-761, Oct. 1996.

BASILI, V.; CALDIERA, G.; ROMBACH, H. D. Measurement - Encyclopedia of software engineering. In: MARCINIAK, J. J. (Org.). **Encyclopedia of software engineering**: volume 2. 2. ed. New York: John Wiley & Sons, 2002. p. 646-661.

BAXTER, G. et al. Understanding the shape of java software. **ACM Sigplan Notices**, New York, v. 41, n. 10, p. 397-412, Oct. 2006.

BEEBE, K. R.; KOWALSKI, B. R. An introduction to multivariate calibration and analysis. **Analytical Chemistry**, Washington, v. 59, n. 17, p. 1007-1017, Sept. 1987.

BENLARBI, S.; MELO, W. L. Polymorphism measures for early risk prediction. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 1999, Los Angeles. **Proceedings…** Los Angeles: [s.n.], 1999. p. 334-344.

BHAT, T.; NAGAPPAN, N. Building scalable failure-proneness models using complexity metrics for large scale software systems. In: ASIA-PACIFIC SOFTWARE ENGINEERING CONFERENCE, 2006, Aichi. **Proceedings…** Aichi: [s.n.], 2006. p. 361-366.

BIEMAN, J.; KANG, B. Cohesion and reuse in an object-oriented system. **ACM Symposium on Software Reusability**, Washington, v. 20, p. 259-262, Aug. 1995.

BREYFOGLE, F. W.; CUPELLO J. M.; MEADOWS, B. **Managing six sigma:** a practical guide to understanding, assessing, and implementing the strategy that yields bottom-line success. New York: John Wiley & Sons, 2001. 288 p.

BRIAND, L. C. et al. Exploring the relationship between design measures and software quality in object-oriented systems. **Journal of Systems and Software**, New York, v. 51, n. 3, p. 245-273, May 2000.

BRIAND, L. C.; DALY, J.; WUEST, J. A unified framework for cohesion measurement in object-oriented systems. **Empirical Software Engineering**, Oxford, v. 3, n. 1, p. 65-117, Mar. 1998.

BRIAND, L.; DEVANBU, P.; MELO, W. An investigation into coupling measures for C++. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 1997, Boston. **Proceedings…** Boston: [s.n.], 1997. p. 412-421.

BROWN, T. A. **Confirmatory factor analysis is for applied research**. New York: The Guilford Press, 2006. 475 p.

BRUNTINK, M.; DEURSEN, A. Predicting class testability using object-oriented metrics. In: INTERNATIONAL WORKSHOP ON SOURCE CODE ANALYSIS AND MANIPULATION, 4., 2004, New York. **Proceedings…** New York: IEEE, 2004. p. 136-145.

BUBSHAIT, A. A. Owner involvement in project quality. **International Journal of Project Management**, Guildford, v. 12, n. 2, p. 115-117, May 1994.

CAMPOS, V. F. **Qualidade total padronização de empresas**. Belo Horizonte: Fundação Christiano Ottoni, 1992. 122 p.

CAPILUPPI, A.; MICHLMAYR, M. From the cathedral to the bazaar: an empirical study of the lifecycle of volunteer community projects. In: FELLER, J. et al. (Ed.). **Open source development, adoption and innovation.** Limerick: Springer, 2007. p. 31-44.

CHELIOTIS, G. From open source to open content: organization, licensing, and decision processes in open cultural production. **Decision Support Systems**, Amsterdam, v. 47, n. 3, p. 229-244, June 2009.

CHIDAMBER, S. R.; DARCY, D. P.; KEMERER, C. F. Managerial use of metrics for object-oriented software: an exploratory analysis. **IEEETransactions on Software Engineering**, New York, v. 24, n. 8, p. 629-639, Aug. 1998.

CHIDAMBER, S.; KEMERER, C. A metrics suite for object-oriented design. **IEEE Transactions on Software Engineering**, New York, v. 20, n. 6, p. 476-493, June 1994.

CHIDAMBER, S.; KEMERER, C. Towards a metrics suite for object-oriented design. **ACM SIGPLAN Notices**, New York, v. 26, n. 11, p. 197-211, Nov. 1991.

CHIN, W. W. The Partial Least Squares Approach for Structural Equation Modeling. In: MARCOULIDES, G. A. (Ed.). **Modern methods for businesss research.** Londres: Lawrence Erlbaum, 1998. p. 295-336.

CHIN, W. W.; DIBBERN, J. An introduction to a permutation based procedure for multi-group PLS analysis: results of tests of differences on simulated data and a cross cultural analysis of the sourcing of information system services between Germany and the USA. In: VINZI, V. E. **Handbook of partial least squares**. Berlin: Springer Handbooks of Computational Statistics, 2009. p. 171-193.

CHIN, W. W.; MARCOLIN, B. L.; NEWSTED, P. R. A partial least squares latent variable modeling approach for measuring interaction effects: results from a Monte Carlo Simulation study and an electronic-mail emotion/adoption study. **Information Systems Research**, Providence, v. 14, n. 2, p. 189-217, June 2003.

CHOWDHURY, I.; ZULKERNINE, M. Can Complexity, Coupling, and Cohesion Metrics be used as Early Indicators of Vulnerabilities? In: ACM SYMPOSIUM ON APPLIED COMPUTING, 2010, Sierre. **Proceedings…** Sierre: [s.n.], 2010. p. 1963-1969.

CHOWDHURY, I.; ZULKERNINE, M. Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities. **Journal Systems Architecture**, Amsterdam, v. 57, n. 3, p. 294-313, Mar. 2011.

CHURCHILL, G. A. A paradigm for developing better measures of marketing constructs. **Journal of Marketing Research**, Chicago, v. 16, n. 1, p. 64-73, Feb. 1979.

CONCAS, G. et al. Assessing traditional and new metrics for object-oriented systems. In: WORKSHOP ON EMERGINGTRENDS IN SOFTWARE METRICS, 2010, Cape Town. **Proceedings…** Cape Town: [s.n.], 2010. p. 24-31.

CORRÊA, U. B. et al. Towards estimating physical properties of embedded systems using software quality metrics. In: INTERNATIONAL CONFERENCE ON COMPUTER AND INFORMATION TECHNOLOGY, 10., 2010, Bradford. **Proceedings…** Bradford: IEEE, 2010. p. 2381-2386.

COUNSELL, S.; SWIFT, S.; CRAMPTON, J. The interpretation and utility of three cohesion metrics for object-oriented design. **ACM Transaction on Software Engineering Methodology**, Amsterdam, v. 15, n. 2, p. 123-149, Apr. 2006.

CRUZ, A. E. C.; OCHIMIZU, K. A. UML approximation of three chidamber-kemerer metrics and their ability to predict faulty code across software projects. **Journal of the IEICE Transactions on Information and Systems**, Japão, v. E93-D, n. 11, p. 3038-3050, Nov. 2010.

DAVISON, A. C.; HINKLEY, D. V. **Bootstrap methods and their application.** Cambridge: Cambridge University Press, 1997. 594 p.

DEKOENIGSBERG, G. How successful open source projects work, and how and why to introduce students to the open source world. In: CONFERENCE ON SOFTWARE ENGINEERING EDUCATION AND TRAINING, 21., 2008, Charleston. **Proceedings…** Charleston: IEEE, 2008. p. 274-276.

EFRON, B.; TIBSHIRANI, R. J. **An introduction to the bootstrap**. New York: Chapman Hall, 1993. 456 p.

EMANUEL, A. W. R. et al. Statistical analysis on software metrics affecting modularity in open-source software. **International Journal of Computer Science & Information Technology**, Oxford, v. 3, n. 3, p. 105-118, June 2011.

ENGLISH, M. et al. Fault detection and prediction in an open-source software project. In: INTERNATIONAL CONFERENCE ON PREDICTOR MODELS IN SOFTWARE ENGINEERING, 5., 2009, Canada. **Proceedings…** Canada: ACM, 2009. p. 1-11.

ENGLISH, M. et al. Measuring the impact of friends on the internal attributes of software systems. In: INTERNATIONAL WORKSHOP ON SOURCE CODE ANALYSIS AND MANIPULATION, 5., 2005, Budapest. **Proceedings…** Budapest: IEEE Computer Society, 2005. p. 151-160.

ENGLISH, M.; BUCKLEY, J.; CAHILL, T. A replicated and refined empirical study of the use of friends in C++ software. **Journal of System and Software,** New York, v. 83, n. 11, p. 2275-2286, Nov. 2010.

ERDIL, K. et al. **Software maintenance as part of the software life cycle, Comp180:** software engineering project. Boston: Department of Computer Science, 2003. 49 p.

ESKI, S.; BUZLUCA, F. An empirical study on object-oriented metrics and software evolution in order to reduce testing costs by predicting change-prone classes. In: INTERNATIONAL CONFERENCE ON SOFTWARE TESTING, VERIFICATION AND VALIDATION, 14., 2011, Berlin. **Proceedings…** Berlin: IEEE, 2011. p. 566-571.

ETZKORN, L. H. et al. A comparison of cohesion metrics for object-oriented systems. **Information and Software Technology**, London, v. 46, n. 10, p. 677-687, Aug. 2004.

FAROOQ, A.; BRAUNGARTEN, R.; DUMKE, R. R. An empirical analysis of object-oriented metrics for java technologies. In: INTERNATIONAL MULTITOPIC CONFERENCE, 9., 2005, Karachi. **Proceedings...** Karachi: IEEE, 2005. p. 1-6.

FERREIRA, D. F. **Estatística multivariada**. Lavras: Editora da UFLA, 2011. 675 p.

FERREIRA, K. A. M. et al. Identifying thresholds for object-oriented software metrics. **Journal of System and Software**, New York, v. 85, n. 2, p. 244-257, Feb. 2012.

FIELD, A. **Discovering statistics using SPSS**. 4. ed. Londres: Sage, 2013. 915 p.

FORNEL
L, C.; LARCKER, D. F. Evaluating structural equation models with unobservable variables and measurement error. **Journal of Marketing Research**, Chicago, v. 18, n. 1, p. 39-50, Feb. 1981.

GAMMA, E. et al. **Padrões de projeto:** soluções reutilizáveis de software orientado a objetos. São Paulo: Bookman. 2000. 364 p.

GATRELL, M.; COUNSELL, S. Size, inheritance, change, and fault-proneness in C# software. **Journal of Object Technology**, Oxford, v. 9, n. 5, p. 29-54, Sept. 2010.

GEFEN, D.; STRAUB, D. A practical guide to factorial validity using pls-graph: tutorial and annotated example. **Communications of the Association for Information Systems**, Elmsford, v. 16, n. 5, p. 91-109, July 2005.

GEFEN, D.; STRAUB, D. W.; BOUDREAU, M. C. Structural equation modeling and regression: guidelines for research practice. **Communications of the Association for Information Systems,** Elmsford, v. 4, n. 7, p. 1-70, Dec. 2000.

GELADI, P.; KOWALSKI, B. R. Partial least-squares regression: a tutorial. **analytica Chimica Acta**, Amsterdam, v. 185, p. 1-17, 1986.

GIL, A. C. **Métodos e técnicas de pesquisa social.** 6. ed. São Paulo: Atlas, 2012. 200 p.

GITHUB. Disponível em: < https://github.com/>. Acesso em: 14 mar. 2015.

GIVEN, L. M. **The sage encyclopedia of qualitative research methods**. Los Angeles: Sage Publications, 2008. 107 p.

GOEL, B. M.; BHATIA, P. K. Analysis of reusability of object-oriented systems using object-oriented metrics. **SIGSOFT Software Engineering Notes,** New York, v. 38, n. 4, p. 1-5, July 2013.

GRANJA-ALVAREZ, J. C.; BARRANCO-GARCÍA, M. J. A method for estimating maintenance cost in a software project: a case study. **Journal of Software Maintenance**, Sussex, v. 9, n. 3, p. 161-175, May/June 1997.

GUPTA, V.; CHHABRA, J. K. Package level cohesion measurement in object-oriented software. **Journal of the Brazilian Computer Society**, Rio de Janeiro, v. 18, n. 3, p. 251-261, Sept. 2012.

GURBANI, V. K.; GARVERT, A.; HERBSLEB, J. D. A case study of open source tools and practices in commercial setting. **ACM SIGSOFT Software Engineering Notes**, New York, v. 30, n. 4, p. 1-6, July 2005.

GYIMOTHY, T.; FERENC, R.; SIKET, I. Empirical Validation of Object-Oriented Metrics on Open Source Software for Fault Prediction. **IEEETransactions on Software Engineering**, New York, v. 31, n.10, p. 897-910, Oct. 2005.

HAIR, J. F. et al. **A Primer on partial least squares structural equation modeling (PLS-SEM).** Thousand Oaks: Sage, 2014. 306 p.

HAIR, J. F. et al. **Análise multivariada de dados**. 6. ed. Porto Alegre: Bookman, 2009. 688 p.

HAIR, J. F.; RINGLE, C. M.; SARSTEDT, M. PLS-SEM: indeed a silver bullet. **Journal of Marketing Theory and Practice**, Oxford, v. 19, n. 2, p. 139-151, 2011.

HALL, T. et al. A systematic literature review on fault prediction performance in software engineering. **IEEE Transactions on Software Engineering**, New York, v. 38, n. 6, p. 1276-1304, Nov. 2012.

HAMID, N. F. I. A.; HASAN, M. K. Industrial-based object-oriented software quality measurement system and its importance. In: INTERNATIONAL SYMPOSIUM IN INFORMATION TECHNOLOGY, 3., 2010, Kuala Lumpur. **Proceedings…** Kuala Lumpur: IEEE, 2010. p. 1332-1336.

HASAN, M. S.; HASAN, M. S. Principal component analysis of coupling measures for developing high quality object oriented software. In: INTERNATIONAL CONFERENCE ON COMPUTER AND COMMUNICATION ENGINEERING, 2010, Kuala Lumpur. **Proceedings…** Kuala Lumpur: IEEE, 2010. p. 1-6, 2010.
HENDERSON-SELLERS, B. **Software metrics**. Hemel Hempstaed: Prentice-Hall, 1996. 234 p.

HENSELER, J.; RINGLE, C. M.; SINKOVICS, R. R. The use of partial least squares path modeling in international marketing. In: ZOU, S. **Advances in international marketing**. Washington: Emerald, 2009. p. 277-319.

HILBURN, T. B.; TOWHIDNEJAD, M. Software quality across the curriculum. **Annual Frontiers in Education**, Elmsford, v. 3, p. S1G-18-S1G-23, 2002.

HITZ, M.; MONTAZERI, B. Measuring coupling and cohesion in object-oriented systems. In: INTERNATIONAL SYMPOSIUM ON APPLIED CORPORATE COMPUTING, 1995, Vienna. **Proceedings…** Vienna: IEEE, 1995.

HONGLEI, T.; WEI, S.; YANAN, Z. The research on software metrics and software complexity metrics. In: INTERNATIONAL FORUM ON

COMPUTER SCIENCE-TECHNOLOGY AND APPLICATIONS, 9., 2009, Chongqing. **Proceedings…** Chongqing: IEEE, 2009. p. 131-136.

HUSEIN, S.; OXLEY, A. A. Coupling and cohesion metrics suite for object-oriented software. In: INTERNATIONAL CONFERENCE ON COMPUTER TECHNOLOGY AND DEVELOPMENT, 9., 2009, Kota Kinabalu. **Proceedings…** Kota Kinabalu: IEEE, 2009. p. 421-425.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. **ISO 15504-1**: information technology - process assessment - part 1 - concepts and vocabulary. Geneva: ISO, 2004.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. **ISO 15939**: systems and software engineering - measurement process. Geneva: ISO, 2007.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. **ISO 24765**: systems and software engineering - vocabulary. Geneva: ISO, 2010.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. **ISO 25010**: systems and software engineering - systems and software quality requirements and evaluation(SQuaRE) - system and software quality models. Geneva: ISO, 2011.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. **ISO 9126-1**: software engineering - product quality - part 1: quality model. Geneva: ISO, 2001.

JALOTE, P. **An integrated approach to software engineering.** 3. ed. New York: Springer Science & Business Media, 2005. 566 p.

JANES, A. et al. Identification of defect-prone classes in telecommunication software systems using design metrics. **Information Sciences**, New York, v. 176, n. 24, p. 3711-3734, Dec. 2006.

JOHNSON, R. A.; WICHERN, D. W. **Applied multivariate statistical analysis.** 6. ed. Rio de Janeiro: Prentice Hall, 2007. 800 p.

JUNG, C. F. **Metodologia científica e tecnológica**. 2. ed. Taquara: FACCAT, 2009.

JUNG, H. W.; KIM, S. G.; CHUNG, C. S. Measuring software product quality: a survey of ISO/IEC 9126. **IEEE Computer Software**, New York, v. 21, n. 5, p. 88-92, Sept./Oct. 2004.

JURAN, J. M.; GRYNA JÚNIOR, F. M. **Quality planning and analysis:** from product development through usage. New York: McGraw-Hill, 1970. 730 p.

KAN, S. H. **Metrics and models in software quality engineering**. 2. ed. Boston: Addison-Wesley Longman, 2002. 560 p.

KANMANI, S. et al. Object-oriented software fault prediction using neural networks. **Information and Software Technology**, London, v. 49, n. 5, p. 483-492, May 2007.

KAUR, A.; KAUR, K. Statistical comparison of modelling methods for software maintainability prediction. **International Journal of Software Engineering and Knowledge Engineering**, Oxford, v. 23, n. 6, p. 743-774, Aug. 2013.

KAUR, K.; SINGH, H. An investigation of design level class cohesion metrics. **The International Arab Journal of Information Technology**, Arábia, v. 9, n. 1, p. 66-73, Jan. 2012.

KAWAGUCHI, S. et al. MUDABlue: an automatic categorization system for open source repositories. ASIA-PACIFIC SOFTWARE ENGINEERING CONFERENCE, 11, 2004, [S.l.], 2004. **Proceedings…** [S.l.]: IEEE, 2004. p. 184-193.

KAYARVIZHY, N.; KANMANI, S. Analysis of quality of object oriented systems using object oriented metrics. In: INTERNATIONAL CONFERENCE ON ELECTRONICS COMPUTER TECHNOLOGY, 3., 2011, Kanyakumari. **Proceedings…** Kanyakumari: IEEE, 2011. p. 203-206.
KEARNEY, J. K. et al. Software complexity measurement. **Communications of the ACM**, New York, v. 29, n. 11, p. 1044-1050, 1986.

KEVREKIDIS, K. et al. Software complexity and testing effectiveness: an empirical study. In: RELIABILITY AND MAINTAINABILITY SYMPOSIUM, 2009, Fort Worth. **Proceedings…** Fort Worth: IEEE, 2009. p. 539-543.

KIEWKANYA, M.; JINDASAWAT, N.; MUENCHAISRI, P. A. Methodology for constructing maintainability model of object-oriented design. In: INTERNATIONAL CONFERENCE ON QUALITY SOFTWARE, 4., 2004, [S.l.]. **Proceedings…** [S.l.], 2004. p. 206-213.

KITCHENHAM, B. Procedures for performing systematic reviews. **Technical Report TR/SE-0401.** Disponível: <http://www.idi.ntnu.no/ emner/empse/papers/kitchenham_2004.pdf>. Acesso em: 29 mar. 2013.

KITCHENHAM, B. A.; WALKER, J. G. A quantitative approach to monitoring software development. **Software Engineering Journal**, London, v. 4, n. 1, p. 2-13, Jan. 1989.

KOSCIANSKI, A.; SOARES, M. dos S. **Qualidade de software.** 2. ed. São Paulo: Novatec, 2006. 395 p.

KPODJEDO, S. et al. Design evolution metrics for defect prediction in object oriented systems. **Empirical Software Engineering**, London, v. 16, n. 1, p. 141-175, Feb. 2011.

KULKARNI, U. L.; KALSHETTY, Y. R.; ARDE, V. G. Validation of CK metrics for object oriented design measurement. In: INTERNATIONAL CONFERENCE ON EMERGING TRENDS IN ENGINEERING AND TECHNOLOGY, 3., 2010, Goa. **Proceedings…** Goa: IEEE, 2010. p. 646-651.

KUMAR, V.; MAHESHWARI, B.; KUMAR, U. An investigation of critical management issues in ERP implementation: empirical evidence from Canadian Organizations. **Technovation**, Essex, v. 23, p. 793- 807, 2003.

KWONG, K.; WONG, K. Partial least squares structural equation modeling (PLS-SEM) techniques using SmartPLS. **Marketing Bulletin**, Washington, v. 24, p. 1-32, 2013.

LAKE, A.; COOK, C. Use of factor analysis to develop OOP software complexity metrics. In: ANNUAL OREGON WORKSHOP ON SOFTWARE METRICS, 1994, Silver Falls. **Proceedings…** Silver Falls: [s.n.], 1994.

LAWRIE, T.; GACEK, C. Issues of dependability in open source software development. **ACM SIGSOFT Software Engineering Notes**, New York, v. 27, n. 3, p. 34-37, May 2002.

LEE, G.; XIA, W. Toward agile: an integrated analysis of quantitative and qualitative field data. **MIS Quarterly,** Minneapolis, v. 34, n. 1, p. 87-114, 2010.

LEE, Y. et al. Measuring the coupling and cohesion of an object-oriented program based on information flow. In: INTERNATIONAL CONFERENCE ON SOFTWARE QUALITY, 1995, [S.l.]. **Proceedings…** [S.l: s.n.], 1995.

LEE, Y.; YANG, J.; CHANG, K. H. Metrics and evolution in open source software. In: INTERNATIONAL CONFERENCE ON QUALITY SOFTWARE, 17., 2007, Portland. **Proceedings…** Portland: IEEE, 2007.p. 191-197.

LEHMAN, M. M.; BELADY, L. A. **Software:** practice and experience. New York: Academic Press, 1985. 538 p.

LI, W.; HENRY, S. Object-oriented metrics that predict maintainability. **Journal of Systems and Software**, New York, v. 23, n. 2, p. 111-122, Nov. 1993.

LORENZ, M.; KIDD, J. **Object-oriented software metrics.** New York: Prentice Hall, 1994. 146 p.

LOWRY, P. B.; GASKIN, J. Partial Least Squares (PLS) Structural Equation Modeling (SEM) for building and testing behavioral causal theory: when to choose it and how to use it. **IEEE Transactions on Professional Communication**, New York, v.57, n. 2, p. 123-146, June 2014.

LU, H. et al. The ability of object-oriented metrics to predict change-proneness: a meta-analysis. **Empirical Software Engineering**, Elmsford, v. 17, n. 3, p. 200-242, June 2012.

MA, Y. et al. A complexity metrics set for large-scale object-oriented software systems. In: INTERNATIONAL CONFERENCE ON COMPUTER AND INFORMATION TECHNOLOGY, 6., 2006, Seoul. **Proceedings…** Seoul: IEEE, 2006. p. 189-189.

MA, Y. et al. A hybrid set of complexity metrics for large-scale object-oriented software systems. **Journal of Computer Science and Technology**, Oxford, v. 25, n. 6, p. 1184-1201, Nov. 2010.

MALHOTRA, R.; KHANNA, M. Investigation of relationship between object-oriented metrics and change proneness. **International Journal of Machine Learning and Cybernetics**, New York, v. 4, n. 4, p. 273-286, Aug. 2013.

MARCONI, M. A.; LAKATOS, E. M. **Fundamentos de metodologia científica**. 7. ed. rev. São Paulo: Atlas, 2010. 315 p.

MAROCO, J. **Análise estatística com a utilização do SPSS.** 3. ed. Lisboa: Silabo, 2010. 822 p.

MARTIN, R. C.; MARTIN, M. **Agile principles, patterns, and practices in C#.** New York: Prentice Hall, 2011. 768 p.

MATA-TOLEDO, R. A.; GUSTAFSON, D. A. A factor analysis of software complexity measures. **Journal of Systems and Software**, New York, v. 17, n. 3, p. 267-273, Mar. 1992.

MAUCZKA, A.; GRECHENIG, T.; BERNHART, M. Predicting code change by using static metrics. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING RESEARCH, MANAGEMENT AND APPLICATIONS, 7., 2009, Haikou. **Proceedings…** Haikou: IEEE, 2009. p. 64-71.

MCCABE, T. J. A complexity measure. **IEEE Transactions on Software Engineering**, New York, v. 2, n. 4, p. 308-320, 1976.

MCMILLAN, C. et al. Categorizing software applications for maintenance. In: INTERNATIONAL CONFERENCE ON SOFTWARE MAINTENANCE, 27., 2011, Williamsburg. **Proceedings…** Williamsburg: IEEE, 2011. p. 343-352.

MEIRELLES, P. et al. A study of the relationships between source code metrics and attractiveness in free software projects. In: BRAZILIAN SYMPOSIUM ON SOFTWARE ENGINEERING, 2010, Salvador. **Proceedings…** Salvador: IEEE, 2010. p. 11-20.

MEYERS, T. M.; BINKLEY, D. An empirical study of slice-based cohesion and coupling metrics. **Transaction Software Engineering Methodology,** New York, v. 17, n. 1, p. 1-27, Dec. 2007.

MIDHA, V.; BHATTACHERJEE, A. Governance practices and software maintenance: a study of open source projects. **Decision Support Systems,** Amsterdam, v. 54, n. 1, p. 23-32, Dec. 2012.

MINGOTI, S. A. **Análise de dados através de métodos de estatística multivariada:** uma abordagem aplicada. Belo Horizonte: Editora da UFMG, 2005. 297 p.

MISHRA, B.; SHUKLA, K. K. Impact of attribute selection on defect proneness prediction in OO software. In: INTERNATIONAL CONFERENCE ON

COMPUTER AND COMMUNICATION TECHNOLOGY, 2., 2011, Allahabad. **Proceedings…** Allahabad: IEEE, 2011. p. 367-372.

MORASCA, S. A probability-based approach for measuring external attributes of software artifacts. In: INTERNATIONAL SYMPOSIUM ON EMPIRICAL SOFTWARE ENGINEERING AND MEASUREMENT, 3., 2009, Lake Buena Vista. **Proceedings…** Lake Buena Vista: IEEE, 2009. p. 44-55.

MUHAMMAD, S.; MAQBOOL, O.; ABBASI, A. Q. Evaluating relationship categories for clustering object-oriented software systems. **IEEE Software**, London, v. 6, n. 3, p. 260-274, June 2012.

MUTHANNA, S. et al. A maintainability model for industrial software systems using design level metrics. In: PROCEEDINGS OF THE SEVENTH WORKING CONFERENCE ON REVERSE ENGINEERING, 7., 2008, Washington. **Proceedings...** Washington: IEEE Computer Society, 2000. p. 248-256.

NAH, F. F.-H.; FAJA, S.; CATA, T. Characteristics of ERP software maintenance: a multiple case study. **Journal of Software Maintenance and Evolution Research and Practice**, v. 13, n. 6, p. 399-414, 2001.

NAPOLI, C.; PAPPALARDO, G.; TRAMONTANA, E. Using modularity metrics to assist move method refactoring of large systems. In: INTERNATIONAL CONFERENCE ON COMPLEX, INTELLIGENT, AND SOFTWARE INTENSIVE SYSTEMS, 17., Catânia. **Proceedings…** Catânia: [s.n.], 2013. p. 529-534.

OFFUTT, A. J.; ABDURAZIK, A.; ALEXANDER, R. T. An analysis tool for coupling-based integration testing. In: INTERNATIONAL CONFERENCE ON ENGINEERING OF COMPLEX COMPUTER SYSTEMS, 6., 2000, Tokyo. **Proceedings…** Tokyo: IEEE, 2000. p. 172-178.

OFFUTT, J.; ABDURAZIK, A.; SCHACH, S.R. Quantitatively measuring object-oriented couplings. **Software Quality Journal,** Essex, v. 16, n. 4, p. 489-512, Dec. 2008.

OLAGUE, H. M. et al. An empirical validation of object-oriented class complexity metrics and their ability to predict error-prone classes in highly iterative, or agile, software: a case study. **Journal of Software Maintenance and Evolution**, Washington, v. 20, n. 3, p. 171-197,May/June 2008.

OLAGUE, H. M. et al. Empirical validation of three software metrics suites to predict fault-proneness of object-oriented classes developed using highly iterative or agile software development processes. **IEEE Transactions on Software Engineering**, New York, v. 33, n. 6, p. 402-419, June 2007.

OLIVEIRA, K. S.; SILVA, D. O.; SOUZA, W. V. de. Barreiras percebidas por médicos do Distrito Federal para a promoção da alimentação saudável. **Cadernos Saúde Coletetiva**, Rio de Janeiro, v. 22, n. 3, p. 260-265, set. 2014.

ORENYI, B. A.; BASRI, S.; JUNG, L. T. Object-oriented software maintainability measurement in the past decade. In: INTERNATIONAL CONFERENCE ON ADVANCED COMPUTER SCIENCE APPLICATIONS AND TECHNOLOGIES, 2012, Kuala Lumpur. **Proceedings…** Kuala Lumpur, 2012. p. 257-262.

PAECH, B.; REUSCHENBACH, B. Open source requirements engineering. In: IEEE INTERNATIONAL REQUIREMENT ENGINEERING CONFERENCE, 14., 2006, Minneapolis. **Proceedings…** Minneapolis: IEEE, 2006. p. 257-262.

PAI, G. J.; DUGAN, J. B. Empirical analysis of software fault content and fault proneness using bayesian methods. **IEEE Transactions on Software Engineering**, New York, v. 33, n. 10, p. 675-686, Sept. 2007.

PANIGRAHI, R.; BABOO, S.; PADHY, N. The statistical measurement of an object-oriented programme using an object oriented metrics. **Advances in Intelligent Systems and Computing**, Amsterdam, v. 328,p. 605-618, 2015.

PARNAS, D. L. On the criteria to be used in decomposing systems into modules. **Communications of the ACM**, New York, v. 15, n. 12, p. 1053-1058, Dec. 1972.

PARTHASARATHY, S.; ANBAZHAGAN, N. Analyzing the software quality metrics for object oriented technology. **Journal of Information Technology**, London, v. 5, n. 6, p. 1053-1057, 2006.

PENDHARKAR, P. C. An exploratory study of object-oriented software component size determinants and the application of regression tree forecasting models. **Information & Management**, Amsterdam, v. 42, n. 1, p. 61-73, Dec. 2004.

PESTANA, M. H.; GAGEIRO, J. N. **Análise de dados para ciências sociais:** a complementaridade do SPSS. 6. ed. Lisboa: Sílabo, 2014. 1240 p.

PFLEEGER, S. L.; ATLEE, J. M. **Software engineering**: theory and practice. 4. ed. Rio de Janeiro: Prentice Hall, 2010. 756 p.

PING, L. A quantitative approach to software maintainability prediction. In: INTERNATIONAL FORUM ON, 2010, Kunming. **Proceedings…** Kunming, 2010. p. 105-108.

PIZZI, N. J. A fuzzy classifier approach to estimating software quality. **Information Sciences**, New York, v. 241, p. 1-11. 2013.

PLOSCH, R. et al. The EMISQ method - expert based evaluation of internal software quality. In: SOFTWARE ENGINEERING WORKSHOP, 31., 2007, Columbia. **Proceedings…** Columbia: IEEE, 2007. p. 99-108.

PMBOK GUIDE. **A guide to the project management body of knowledge.** 5. ed. New York: Project Management Institute, 2013. 589p.

PRASAD, L.; NAGAR, A. Experimental analysis of different metrics (object-oriented and structural) of software. In: INTERNATIONAL CONFERENCE ON COMPUTATIONAL INTELLIGENCE, COMMUNICATION SYSTEMS AND NETWORKS, 1., 2009, Indore. **Proceedings…** Indore: IEEE, 2009. p. 235-240.

PRESSMAN, R. S. **Software engineering:** a practitioner's approach. 8. ed. New York: McGraw-Hill Science, 2014. 976 p.

QUALIPSO. Disponível em: <http://www.qualipso.org>. Acesso em: 26 jul. 2014.

RAJNISH, K.; BHATTACHERJEE, V. Class inheritance metrics-an analytical and empirical approach. **Journal of Computer Science**, Lavras, v. 7, n. 3, p. 25-34, set. 2008.

RATHORE, S. S.; GUPTA, A. Investigating object-oriented design metrics to predict fault-proneness of software modules. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 6., 2012, Indore. **Proceedings…** Indore: IEEE, 2012. p. 1-10.

RATHORE, S. S.; GUPTA, A. Validating the effectiveness of object-oriented metrics over multiple releases for predicting fault proneness. In: ASIA-PACIFIC

SOFTWARE ENGINEERING CONFERENCE, 19., 2012, Hong Kong. **Proceedings…** Hong Kong, 2012. p. 350-355.

RAYKOV, T.; MARCOULIDES, G. A. **A first course in structural equation modeling.** 2. ed. Lawrence: Lawrence Erlbaum Associates, 2012. 248 p.

ROMANO, D.; PINZGER, M. Using source code metrics to predict change-prone java interfaces. In: INTERNATIONAL CONFERENCE ON SOFTWARE MAINTENANCE, 27., 2011, Williamsburg. **Proceedings…** Williamsburg: IEEE, 2011. p. 303-312.

ROUBTSOV, S.; SEREBRENIK, A.; VAN DEN BRAND, M. Dn-based design quality comparison of industrial java applications. In: CENTRAL AND EASTERN EUROPEAN SOFTWARE ENGINEERING CONFERENCE IN RUSSIA, 5., 2009, Moscow. **Proceedings…** Moscow: IEEE, 2009. p. 95-101.

RUNESON, P.; ISACSSON, P. Software quality assurance-concepts and misconceptions. In: EUROMICRO CONFERENCE, 24., 1998, Vasteras. **Proceedings…** Vasteras: IEEE, 1998. p. 853-859.

SAINI, M.; KAUR, K. A review of open source software development life cycle models. **International Journal of Software Engineering and its Applications,** Tasmania, v. 8, n. 3, p. 417-434, 2014.

SAMPIERI, R. H.; COLLADO, C. F.; LUCIO, M. D. P. B. **Metodologia de pesquisa.** 5. ed. São Paulo: Penso, 2013. 624 p.

SANTOS, M. et al. Similar characteristics of internal software quality attributes for object-oriented open-source software projects. In: SIMPÓSIO BRASILEIRO DE QUALIDADE DE SOFTWARE, 13., 2014, Blumenau. **Anais…** Blumenau: [S.l.], 2014. p. 209-223.

SCACCHI, W. et al. Understanding free/open source software development processes. **Software Process Improvement and Practice**, Chichester, v. 11, p. 95-105, 2006.

SCACCHI, W. Understanding the requirements for developing open source software systems. **IEE Software**, Califórnia, v. 149, n. 1, p. 24-39, Aug. 2002.

SCHROEDER, M. A practical guide to object-oriented metrics.**IT Professional**, New York, v. 1, n. 6, p. 30-36, Nov. 1999.

SEBER, G. A. F. **Multivariate observations**. Hoboken: Wiley-Interscience, 2004. 686 p.

SELIYA, N.; KHOSHGOFTAAR, T. M. Software quality analysis of unlabeled program modules with semi supervised clustering. **IEEE Transactions on Systems, Man and Cybernetics, Part A, Systems and Humans**, New York, v. 37, n. 2, p. 201-211, 2007.

SHAHEEN, M. R.; DU BOUSQUET, L. Is depth of inheritance tree a good cost prediction for branch coverage testing? In: INTERNATIONAL CONFERENCE ON ADVANCES IN SYSTEM TESTING AND VALIDATION LIFECYCLE, 1., 2009, Porto. **Proceedings…** Porto: IEEE, 2009. p. 42-47.

SHAHEEN, M. R.; SHAHEEN, M. R. Relation between depth of inheritance tree and number of methods to test. In: INTERNATIONAL CONFERENCE ON SOFTWARE TESTING, VERIFICATION, AND VALIDATION, 1., 2008, Lillehammer. **Proceedings…** Lillehammer: IEEEE, 2008. p. 161-170.

SHARAFAT, A. R.; TAHVILDARI, L. A probabilistic approach to predict changes in object-oriented software systems. In: EUROPEAN CONFERENCE ON SOFTWARE MAINTENANCE AND REENGINEERING,11., 2007, Amsterdam. **Proceedings…** Amsterdam: IEEE, 2007. p. 27-38.

SHARAFAT, A. R.; TAHVILDARI, L. Change prediction in object-oriented software systems: a probabilistic approach. **Journal of Software,** Sussex, v. 3, n. 5, p. 26-39, May 2008.

SHATNAWI, R. et al. Finding software metrics threshold values using ROC curves. **Journal of Software Maintenance and Evolution**, Washington, v. 22, n. 1, p. 1-16, Jan. 2010.

SHATNAWI, R.; ALZU'BI, A. An empirical verification of software artifacts using software metrics. In: INTERNATIONAL MULTICONFERENCE OF ENGINEERS AND COMPUTER SCIENTISTS, 2011, Hong Kong. **Proceedings…** Hong Kong: IMECS, 2011.

SHI, H.; ZHAO, F. Exploring critical factors (CFs) of ERP post-implementations. **Pacific Science Review**, London, v. 11, n. 2, p. 203-209, 2009.

SHIN, Y. et al. Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities. **IEEE Transactions on Software Engineering**, New York, v. 37, n. 6, p.772-787, Nov./Dec. 2011.

SILVA, E. L.; MENEZES, E. M. **Metodologia da pesquisa e elaboração de dissertação**. Florianopolis: Editora da UFSC, 2000. 139 p.

SIMON, H. A. On a class of skew distribution functions. **Biometrika,** London, v. 42, n. 3-4, p. 425-440, Dec. 1955.

SINGH, B.; KANNOJIA, S. P. A review on software quality models. In: INTERNATIONAL CONFERENCE ON COMMUNICATION SYSTEMS AND NETWORK TECHNOLOGIES, Gwalior, 2013. **Proceedings…** Gwalior: IEEE, 2013. p. 801-806.

SINGH, P.; VERNA, S. Empirical investigation of fault prediction capability of object oriented metrics of open source software. In: INTERNATIONAL JOINT CONFERENCE ON COMPUTER SCIENCE AND SOFTWARE ENGINEERING, 2012, Bangkok. **Proceedings…** Bangkok, 2012. p. 323-327.

SINGH, Y.; KAUR, A.; MALHOTRA, R. Empirical validation of object-oriented metrics for predicting fault proneness models. **Software Quality Journal,** Essex, v. 18, n. 1, p. 3-35, Mar. 2010.

SOFTEX. **O impacto do software livre e de código aberto na indústria de software do Brasil**. Campinas: Softex, 2005. 76 p.

SOFTEX. **Qualidade - MPS-BR**. Disponível em: <http://www.softex.br/mpsbr/>. Acesso em 01 nov. 2013.

SOFTWARE ENGINEERING INSTITUTE. **CMMI for development (CMMI-DEV) version 1.3**. Pittsburgh: Software Engineering Institute, Carnegie Mellon University, 2010.

SOFTWARE ENGINEERING STANDARDS COMMITTEEOF THEIEEE COMPUTER SOCIETY. **IEEE standard for developing software life cycle processes.** Piscataway: IEEE-SA Standards Board, 2006. 88 p.

SOMMERVILLE, I. **Software engineering.** 10. ed. São Paulo: Pearson, 2015. 816 p.

SOTO, M.; CIOLKOWSKI, M. The QualOSS open source assessment model measuring the performance of open source communities. In: INTERNATIONAL SYMPOSIUM ON EMPIRICAL SOFTWARE ENGINEERING AND MEASUREMENT, 3., 2009, Lake Buena Vista. **Proceedings…** Lake Buena Vista: IEEE, 2009. p. 498-501.

SOURCEFORGE. Disponível em: <http://sourceforge.net/>. Acesso em: 14 mar. 2015.

SOUZA, L. B. L. de; MAIA, M. de A. Do software categories impact coupling metrics? In: WORKING CONFERENCE ON MINING SOFTWARE REPOSITORIES, 13., 2013, Piscataway. **Proceedings…** Piscataway: IEEE, 2013. p. 217-220.

STEINMACHER, I. et al. Preliminary empirical identification of barriers faced by newcomers to open source software projects. In: BRAZILIAN SYMPOSIUM ON SOFTWARE ENGINEERING, 2014, Maceió. **Proceedings…** Maceió: IEEE, 2014. p. 51-60.

STRAUB, D. W. Validating instruments in MIS research. **MIS Quarterly,** Minneapolis, v. 13, n. 2, p. 147-169, June 1989.

SUCCI, G. et al. An empirical exploration of the distributions of the chidamber and kemerer object-oriented metrics suite. **Empirical Software Engineering**, Sussex, v. 10, n. 1, p. 81-104, Jan. 2005.

SUNDAY, D. A. Software maintainability - a new ‘ility'. In: RELIABILITY AND MAINTAINABILITY SYMPOSIUM, 1989, Atlanta. **Proceedings…** Atlanta: IEEE, 1989. p. 50-51.

TEGARDEN, D.; SHEETZ, S.; MONARCHI, D. A software complexity model of object-oriented systems. **Decision Support Systems,** Amsterdam, v. 13, n. 3-4, p. 241-262, Mar. 1992.

TIAN, Y.; CHEN, C.; ZHANG, C. AODE for source code metrics for improved software maintainability. In: INTERNATIONAL CONFERENCE ON SEMANTICS, KNOWLEDGE AND GRID, 4., 2008, Beijing. **Proceedings…** Beijing: IEEE, 2008. p. 330-335.

TOBIAS, R. D. **An introduction to partial least squares regression.** New York: SUGI, 1995. 8 p.

TONELLI, R. et al. An analysis of SNA metrics on the java qualitas corpus. In: INDIA SOFTWARE ENGINEERING CONFERENCE, 4., 2011, New York. **Proceedings…** New York: ISEC, 2011. p. 205-213.

TREIBLMAIER, H.; FILZMOSER, P. Exploratory factor analysis revisited: how robust methods support the detection of hidden multivariate data structures in IS research. **Information & Management**, Amsterdam, v. 47, n. 4, p. 197-207, May 2010.

URBACH, N.; AHLEMANN, F. Structural equation modeling in information systems research using partial least squares.**Journal of Information Technology Theory and Application,** Amsterdam, v. 11, n. 2, p. 5-40, June 2010.

VAN SOLINGEN, R.; BERGHOUT, E. **The goal/question/metric method:** a practical guide for quality improvement of software development. Rio de Janeiro: McGraw-Hill, 1999. 200 p.

VIRANI, S. et al. Investigation of domain effects on software. In: ANNUAL SOUTHEAST REGIONAL CONFERENCE, 47., 2009, New York. **Proceedings…** New York: ACM, 2009. p. 1-5.

VON HIPPEL, E.; VON KROGH, G. Open source software and the "private-collective" innovation model: issues for organization science. **Organization Science**, Providence, v. 14, n. 2, p. 209-223, Mar. 2003.

WERTS, C. E.; LINN, R. L.; JORESKOG, K. G. Intraclass reliability estimates: testing structural assumptions. **Educational and Psychological Measurement**, Durham, v. 34, n. 1, p. 25-33, Apr. 1974.

XIAO, L. An empirical study of source level complexity. In: INTERNATIONAL CONFERENCE ON COMPUTATIONAL AND INFORMATION SCIENCES, 15., 2013, Shiyang. **Proceedings…** Shiyang: IEEE, 2013. p. 1991-1994.

XU, J.; HO, D.; CAPRETZ, L. An empirical validation of object-oriented design metrics for fault prediction. **Journal of Computer Science**, Lavras, v. 4, n. 7, p. 583-589, 2008.

YIN, R. K. **Estudo de caso:** planejamento e métodos. 4. ed. São Paulo: Bookman, 2010. 248 p.

ZHANG, H.; BABAR, M. A. An empirical investigation of systematic reviews in software engineering. In: INTERNATIONAL SYMPOSIUM ON EMPIRICAL SOFTWARE ENGINEERING AND MEASUREMENT, Banff, 2011. **Proceedings…** Banff: IEEE, 2011. p. 87-96.

ZHANG, H.; TAN, H. B. K. An empirical study of class sizes for large java systems. In: ASIA-PACIFIC SOFTWARE ENGINEERING CONFERENCE,14., 2007, Aichi. **Proceedings…** Aichi: IEEE, 2007. p. 230-237.

ZHONG, H.; YANG, Y.; KEUNG, J. Assessing the representativeness of open source projects in empirical software engineering studies. In: ASIA-PACIFIC SOFTWARE ENGINEERING CONFERENCE, 19., 2012, Hong Kong. **Proceedings…** Hong Kong: IEEE, 2012. p. 808-817.

ZHOU, Y.; LEUNG, H. Empirical analysis of object-oriented design metrics for predicting high and low severity faults. **IEEE Transactions on Software Engineering**, New York, v. 32, n. 10, p. 771-789, Oct. 2006.
ZHOU, Y.; LEUNG, H. Predicting object-oriented software maintainability using multivariate adaptive regression splines. **Journal of Systems and Software**, New York, v. 80, n. 8, p. 1349-1361, Aug. 2007.

ZHOU, Y.; LEUNG, H.; XU, B. Examining the potentially confounding effect of class size on the associations between object-oriented metrics and change-proneness. **IEEE Transactions on Software Engineering**, New York, v. 35, n. 5, p. 607-623, Sept./Oct. 2009.

ZHOU, Y.; XU, B. Predicting the maintainability of open source software using design metrics. **Wuhan University Journal of Natural Sciences**, Wuhan, v. 13, n. 1, p. 14-21, Feb. 2008.