LUANA ALMEIDA MARTINS

ANÁLISE DA MODULARIDADE DE CARACTERÍSTICAS ASPECTUAIS DE TECNOLOGIAS PARA IMPLEMENTAR LINHAS DE PRODUTOS DE SOFTWARE

LAVRAS-MG
2019

LUANA ALMEIDA MARTINS

ANÁLISE DA MODULARIDADE DE CARACTERÍSTICAS ASPECTUAIS DE TECNOLOGIAS PARA IMPLEMENTAR LINHAS DE PRODUTOS DE SOFTWARE

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

Prof. Dr. Heitor Augustus Xavier Costa
Orientador

Prof. Dr. André Pimenta Freire
Coorientador

LAVRAS-MG
2019

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

Martins, Luana Almeida.
Análise da Modularidade de Características Aspectuais de Tecnologias para Implementar Linhas de Produtos de Software / Luana Almeida Martins. - 2019.
159 p. : il.

Orientador(a): Heitor Augustus Xavier Costa.
Coorientador(a): André Pimenta Freire.
Dissertação (mestrado acadêmico) - Universidade Federal de Lavras, 2019.
Bibliografia.

1. Linha de Produtos de Software. 2. Modularidade. 3. Qualidade de Software. I. Costa, Heitor Augustus Xavier. II. Freire, André Pimenta. III. Título.

LUANA ALMEIDA MARTINS

ANÁLISE DA MODULARIDADE DE CARACTERÍSTICAS ASPECTUAIS DE TECNOLOGIAS PARA IMPLEMENTAR LINHAS DE PRODUTOS DE SOFTWARE

MODULARITY ANALYSIS OF ASPECTUAL FEATURE MODULES OF TECHNOLOGIES TO SOFTWARE PRODUCT LINES DEVELOPMENT

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

APROVADA em 29 de agosto de 2019

Dr. Paulo Afonso Parreira Júnior UFLA
Dr. Ivan do Carmo Machado UFBA

Prof. Dr. Heitor Augustus Xavier Costa
Orientador

Prof. Dr. André Pimenta Freire
Coorientador

LAVRAS-MG
2019

AGRADECIMENTOS

Gostaria de agradecer à Universidade Federal de Lavras, especialmente ao Departamento de Ciência da Computação, pela oportunidade de obter meu mestrado. Ademais, o presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES).

Gostaria de agradecer aos professores Dr. Heitor Costa e Dr. André Pimenta Freire, pela paciência, motivação e orientação durante a realização deste trabalho de mestrado. Além disso, gostaria de agradecer ao restante do meu comitê de dissertação, o Dr. Paulo Afonso Parreira Júnior e Ivan do Carmo Machado, pela leitura cuidadosa deste texto e pelos seus comentários e sugestões de melhoria.

RESUMO

Linha de Produtos de Software (LPS) visa à geração de produtos de software pertencentes a determinado domínio, por meio do reúso sistemático de artefatos de software. Para tanto, a abordagem consiste na identificação de *features* comuns, permitem que os produtos sejam desenvolvidos com comportamento padrão, e *features* variáveis, permitem a variação no comportamento dos produtos e compõem os produtos específicos. O ponto crítico da LPS é a modularização dos artefatos de software para serem coerentes, bem definidos, independentes e combináveis. Para isso, são utilizadas tecnologias de gerenciamento de variabilidades que permitem o desenvolvimento do conjunto de *features* da LPS e suportam configuradores para a propagação de escolhas das *features* e a auto-conclusão dos produtos da LPS. Este trabalho propõe uma investigação da modularidade de uma LPS desenvolvida com três tecnologias de gerenciamento de variabilidades: (i) Orientação a Aspectos, (ii) Orientação a Características, e (iii) Módulos de Características Aspectuais. Para essa investigação, foram coletadas três medidas de acoplamento (*Dependency In*, *Dependency Out* e *Structural Feature Coupling*) e três medidas de coesão (*External-ratio Feature Dependency*, *Internal-ratio Feature Dependency* e *Lack of Concern-based Cohesion*). Essas medidas foram analisadas utilizando o teste de significância de Friedman (com nível de 5% de significância). Primeiramente, foi realizada a análise individual das medidas, por meio da qual foi constatada a diferença significativa entre as tecnologias Orientação a Aspectos e Módulos de características Aspectuais, para todas as medidas. Posteriormente, as medidas foram analisadas conforme o agrupamento em medidas de acoplamento e em medidas de coesão. No entanto, não foi constatada diferença significativa entre as três tecnologias de gerenciamento de variabilidades utilizando as medidas agrupadas em acoplamento e em coesão.

Palavras-chave: Linha de Produtos de Software. Modularidade. Qualidade de Software.

ABSTRACT

Software Product Lines (SPL) aim to generate software products belonging to a specific domain, by means of the systematic reuse of software artifacts. Thence, the approach identifies common features, which allows the development of products with standard behavior; and variable features, which allows the variation in product behavior and the composition of specific products. The critical point of SPL is the modularization of software artifacts to be coherent, well-defined, independent, and combinable. For this, the feature set development uses variability management technologies, which support configurators for the automated generation of configurations and derivation of the LPS products. In this work, we proposed to investigate the modularity of an SLP developed with three variability management technologies: i) Aspect Oriented; ii) Feature Oriented; and iii) Aspectual Feature Modules. Thus, we selected three coupling metrics (Dependency In, Dependency Out and Structural Feature Coupling) and three cohesion metrics (External-ratio Feature Dependency, Internal-ratio Feature Dependency and Lack of Concern-based Cohesion). We analyzed these metrics using the Friedman significance test (with a significance level of 5%). Firstly, we analyzed which metric individually, and as a result, there was a significant difference between the technologies Aspect Oriented and Aspectual Feature Modules, for all metrics. After, we analyzed the set of coupling metrics and the set of cohesion metrics. However, there was no significant difference among the three variability management technologies, using the set of cohesion and coupling metrics.

Keywords: Software Product Line. Modularity. Software Quality.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE CÓDIGOS

LISTA DE ACRÔNIMOS

ACM	*Association for Computing Machinery*
AFFOgaTO	*dAtaset For the Feature mOdel evoluTiOn*
AHEAD	*Algebraic Hierarchical Equations for Application Design*
API	*Application Programming Interface*
AST	*Abstract Syntax Tree*
CIDE	*Colored Integrated Development Environment*
CMMI	*Capability Maturity Model Integration*
CSS	*Cascading Style Sheets*
DepIn	*Dependency In*
DepOut	*Dependency Out*
DOM	*Document Object Model*
EFD	*External-ratio Feature Dependency*
ESPREssO	*mEasures dataSet for dynamic sPl featuRE mOdel*
FM	*Feature Model*
FODA	*Feature Oriented Domain Analysis*
HTML	*HyperText Markup Language*
IDE	*Integrated Development Environment*
IEC	*International Electrotechnical Commission*
IEEE	*Institute of Electrical and Electronic Engineers*
IFD	*Internal-ratio Feature Dependency*
ISF	*Italiana Informatici Senza Frontiere*
ISO	*International Organization for Software Standardization*
JDT	*Java Development Tools*
LCC	*Lack of Concern-based Cohesion*
LPS	Linha de Produtos de Software
MAcchiATO	*MeAsures dATaset for feaTure mOdel*
MCA	Módulos de Características Aspectuais
MPS.BR	Melhoria de Processos do Software Brasileiro
OA	Orientação a Aspectos
OC	Orientação a Características
ONG	Organização Não-Governamental
OO	Orientação a Objetos
PCA	*Principal Component Analysis*
SEI	*Software Engineering Institute*
SFC	*Structural Feature Coupling*
SPLiME	*Software Product Line Maintainability Evaluation*
S.P.L.O.T.	*Software Product Lines Online Tools*
SWT	*Standard Widget Toolkit*
TA	Tecnologia Assistiva
W3C	*World Wide Web Consortium*

SUMÁRIO

1 INTRODUÇÃO

A abordagem Linhas de Produtos de Software (LPS) visa à geração de produtos de software pertencentes a determinado domínio, por meio do reúso sistemático de artefatos de software (NORTHROP; CLEMENTS, 2012). Para isso, essa abordagem consiste na identificação de características (*features*) comuns e variáveis entre os produtos do domínio. As *features* comuns permitem que os produtos sejam desenvolvidos com comportamento padrão e as *features* variáveis permitem a variação no comportamento dos produtos e compõem os produtos específicos (APEL et al., 2013). Dessa forma, uma LPS é um conjunto de *features* que podem ser (re)combinadas para a derivação de seus produtos. O ponto crítico da LPS é a manipulação das partes variáveis dos produtos gerados, ou seja, a modularização dos artefatos de software para serem coerentes, bem definidos, independentes e combináveis (BOUCHER et al., 2010; REIS et al., 2014). Particularmente, a modularidade de um sistema de software está relacionada ao acoplamento e à coesão entre suas unidades de decomposição, como as classes e os pacotes, que devem possuir alta coesão e baixo acoplamento (BAVOTA et al., 2010).

Para apoiar a manipulação das partes variáveis da LPS, são utilizadas tecnologias de gerenciamento de variabilidades que permitem o desenvolvimento da LPS em termos do conjunto de features e suas relações (SILVA et al., 2011). Dessa forma, ao modularizar a LPS, espera-se que a manutenção de seu código fonte seja mais fácil, pois as modificações feitas em um módulo da LPS (em uma *feature*) impactam o mínimo possível o restante da LPS. Além disso, quando utilizadas em ambientes específicos para o desenvolvimento de LPS, as tecnologias de gerenciamento de variabilidades suportam configuradores para a propagação de escolhas e a auto conclusão na derivação dos produtos da LPS (ARCAINI et al., 2017). Dessa forma, a modularidade da LPS também impacta na facilidade de derivação de produtos, visto que features com código mais independentes do restante do código da LPS podem demandar menos modificações para serem combinadas.

Diante da complexidade em realizar manutenções em LPS, dado que as alterações feitas em uma unidade de decomposição podem impactar muitos de seus produtos, diversos trabalhos na literatura investigam a importância das tecnologias de gerenciamento de variabilidades para a manutenibilidade da LPS. No geral, esses trabalhos investigam a manutenibilidade em termos da propagação de mudanças e da estabilidade da LPS (FERREIRA et al., 2014; GAIA et al., 2012), ou em relação às propriedades de sistemas software, tais como, acoplamento, coesão, tamanho e complexidade (REIS et al., 2014). Dentre as tecnologias de gerenciamento de

variabilidades investigadas nesses trabalhos, destacam-se as tecnologias baseadas em composição, em que a implementação das features é feita em módulos distintos, e as tecnologias baseadas em anotação, em que as features são marcadas no código sem que sejam fisicamente separadas (REIS et al., 2014; VALE, 2013). Desse modo, quando as tecnologias baseadas em composição e em anotação são comparadas (FERREIRA et al., 2014; GAIA et al., 2012), as tecnologias baseadas em composição apresentam mais estabilidade e facilidade de modificação, pois as features são implementadas em unidades modulares distintas. Diferentemente, ao comparar tecnologias de um mesmo tipo (*e.g.*, tecnologias baseadas em composição (REIS et al., 2014), são analisadas propriedades como o tamanho, em que a verbosidade da linguagem de programação utilizada impacta a manutenibilidade, podendo não trazer dados significativos quanto à modularidade das *features*.

Dentre as tecnologias de gerenciamento de variabilidades analisadas nesses trabalhos, destacam-se Orientação a Aspectos, Orientação a Características e Módulos de Características Aspectuais. Apesar dessas tecnologias serem analisadas em um ou mais trabalhos, elas não foram analisadas em conjunto, em relação a modularização das *features* da LPS. Desse modo, neste trabalho, é proposta uma investigação sobre a modularidade de uma LPS desenvolvida com essas três tecnologias de gerenciamento de variabilidades. A modularidade é investigada em relação ao acoplamento e à coesão entre suas *features*, utilizando medidas específicas para o contexto de LPS. Para mensurar o acoplamento, foram utilizadas as medidas *Dependency In*, *Dependency Out* e *Structural Feature Coupling*. Para mensurar a coesão, foram utilizadas as medidas *External-ratio Feature Dependency*, *Internal-ratio Feature Dependency* e *Lack of Concern-based Cohesion*.

Além disso, para a análise da modularidade, foi desenvolvida uma LPS para sistemas de software de Tecnologia Assistiva (TA). Esses sistemas são utilizados por pessoas com deficiência ou com dificuldades de aprendizagem para superar/reduzir suas limitações funcionais, permitindo mais autonomia e mais independência na realização de suas tarefas. Porém, a utilização desses sistemas, muitas vezes, é limitada pelo alto custo de aquisição e pela dificuldade em encontrar produtos apropriados às necessidades funcionais de cada pessoa (CGEE, 2012). Nesse contexto, a abordagem LPS pode favorecer o desenvolvimento de sistemas de software de TA, com funções estritamente necessárias a cada pessoa, o que pode ocasionar na redução do preço de aquisição do sistema (CALEFATO et al., 2015).

1.1 Motivação

As tecnologias de gerenciamento de variabilidades são importantes para o reúso efetivo de artefatos de software durante o desenvolvimento e a manutenção de uma LPS. Esse reúso é indicado pela capacidade em realizar alterações na LPS sem provocar sua instabilidade. Por exemplo, durante a derivação de produtos, uma *feature* variável pode ser selecionada para compor o produto final. Portanto, os artefatos de software referentes a ela são adicionados ao produto final; caso contrário, eles são removidos. Ao realizar alterações nesses artefatos, a não modularidade das *features* pode ocasionar alterações em cascata por causa das dependências estabelecidas entre elas (GAIA, 2013). Dessa forma, em uma LPS modular, há mais estabilidade, facilitando a manutenibilidade e a reusabilidade de seus artefatos (COLANZI, 2012). Diante da importância da modularidade da LPS para a sua manutenibilidade e reúso de seus artefatos de software, neste trabalho, é investigado se existe diferença significativa entre a modularidade de uma LPS implementada com tecnologias composicionais para o gerenciamento de variabilidades. Para isso, a modularidade é analisada em relação ao acoplamento e à coesão, pois são propriedades frequentemente utilizadas para aferir a modularidade das unidades de decomposição (em LPS, são as *features*) (BAVOTA et al., 2010; BAVOTA et al., 2012; BECK; DIEHL, 2011; COLANZI, 2012).

1.2 Objetivo

Neste trabalho, o objetivo é investigar três tecnologias composicionais (Orientação a Aspectos, Orientação a Características e Módulos de Características Aspectuais) para o gerenciamento de variabilidades, a fim de identificar se existe diferença significativa entre essas tecnologias para a modularidade da LPS quanto ao acoplamento e à coesão das *features*. Para alcançar esse objetivo, foram estabelecidos os seguintes objetivos específicos:

- Identificar como a LPS é utilizada para o desenvolvimento de sistemas de software de TA, por meio de uma revisão da literatura *ad hoc* para definir um estudo de caso que contribua para o estado da arte;
- Desenvolver três versões da LPS definida como estudo de caso, utilizando as seguintes tecnologias de gerenciamento de variabilidades: i) Orientação a Aspectos, utilizando a linguagem AspectJ; ii) Orientação a Características, utilizando a linguagem Jakarta; e iii) Módulos de Características Aspectuais, utilizando a combinação das linguagens Jakarta e AspectJ;

- Identificar medidas de software frequentemente utilizadas em LPS para mensurar as propriedades de acoplamento e de coesão, por meio de um mapeamento sistemático da literatura;
- Desenvolver uma ferramenta computacional (*plug-in* para IDE Eclipse) para coletar as medidas de acoplamento e de coesão em LPS desenvolvida com as três tecnologias de gerenciamento de variabilidades;
- Verificar se existe diferença significativa entre a modularidade da LPS desenvolvida com as três tecnologias de gerenciamento de variabilidades.

1.3 Organização do Trabalho

O trabalho está estruturado da seguinte forma.

No Capítulo 2, são relatados a classificação e o método de pesquisa utilizados.

No Capítulo 3, é apresentada a abordagem LPS, na qual são destacados conceitos, técnicas de desenvolvimento utilizadas, vantagens e desvantagens em relação ao desenvolvimento de software tradicional. Além disso, são apresentadas as tecnologias de gerenciamento de variabilidades que podem ser utilizadas para o desenvolvimento de LPS.

No Capítulo 4, é apresentado o Mapeamento Sistemático da Literatura realizado para identificar as medidas de software frequentemente utilizadas no contexto de LPS.

No Capítulo 5, é apresentado o desenvolvimento do apoio computacional para a coleta das medidas de software em LPS desenvolvidas com as linguagens Jakarta e AspectJ.

No Capítulo 6, é apresentada a avaliação da abordagem, na qual são realizados os processos da engenharia de domínio para a construção da LPS, utilizando as tecnologias Orientação a Aspectos, Orientação a Características e Módulos de Características Aspectuais. Adicionalmente, é apresentada a análise estatística, que consiste na coleta e na análise das medidas de software em cada tecnologia para verificar se há diferença significativa entre elas.

No Capítulo 7, são apresentados os trabalhos relacionados à avaliação das características de qualidade em LPS e à utilização de LPS para o desenvolvimento de recursos de TA.

No Capítulo 8, são relatadas possíveis ameaças a validade do trabalho, classificadas em validade interna, validade externa, validade de construção e validade de conclusão.

No Capítulo 9, são apresentadas as considerações finais do trabalho, nas quais são destacadas as conclusões, as contribuições e as sugestões de trabalhos futuros.

No APÊNDICE A, são apresentados os dados referentes aos artigos selecionados no Mapeamento Sistemático da Literatura.

No APÊNDICE B , são apresentados os dados extraídos dos artigos selecionados no Mapeamento Sistemático da Literatura.

No APÊNDICE C, são apresentados os valores das medidas de software, coletadas durante a realização da análise estatística para as tecnologias de gerenciamento de variabilidades utilizadas no desenvolvimento da LPS.

No APÊNDICE D, são apresentados os valores normalizados das medidas de software para serem utilizados nos testes estatísticos.

No APÊNDICE E, são apresentados os valores das medidas de software agrupados nas propriedades de acoplamento e de coesão de software para as tecnologias utilizadas no desenvolvimento da LPS.

2 METODOLOGIA DE PESQUISA

2.1 Considerações Iniciais

Metodologia de pesquisa é o conjunto de métodos, de técnicas e de procedimentos a serem empregados de modo a viabilizar a execução da pesquisa, a fim de obter como resultado um novo produto, processo ou conhecimento (JUNG, 2009; PRODANOV; FREITAS, 2013).

O restante deste capítulo está organizado da seguinte forma. Na Seção 2.2, a pesquisa é classificada. Na Seção 2.3, as etapas de desenvolvimento da pesquisa são apresentadas.

2.2 Classificação da Pesquisa

As pesquisas podem ser classificadas do ponto de vista de (JUNG, 2009; PRODANOV; FREITAS, 2013).

- **Sua natureza**. Uma pesquisa pode ser classificada em (a) básica, que objetiva gerar conhecimentos novos úteis para o avanço da ciência, sem aplicação prática prevista, ou (b) aplicada, que objetiva gerar conhecimentos para aplicação prática, dirigidos à solução de problemas específicos. Este trabalho pode ser classificado como **pesquisa aplicada**, pois é investigado o valor de medidas de acoplamento e de coesão para a modularidade de uma LPS desenvolvida com três tecnologias de gerenciamento de variabilidades;
- **Seus objetivos**. Uma pesquisa pode ser classificada como (a) exploratória, quando a pesquisa se encontra na fase preliminar e objetiva proporcionar mais informações sobre o assunto, (b) descritiva, na qual o pesquisador apenas registra e descreve os fatos observados sem interferir neles, ou (c) explicativa, em que o pesquisador procura por fatores que contribuem ou determinam a ocorrência de um evento. Este trabalho pode ser classificado como **pesquisa descritiva**, pois objetiva investigar se existe diferença significativa estatisticamente entre a modularidade da LPS desenvolvida com três tecnologias de gerenciamento de variabilidades;
- **Procedimentos técnicos**. Uma pesquisa pode ser classificada em (a) bibliográfica, elaborada a partir de materiais publicados sobre determinado assunto, (b) documental, elaborada a partir de estudos que ainda não receberam tratamento analítico ou podem ser reelaborados de acordo com os objetivos da pesquisa, (c) experimental, são selecionadas variáveis que podem influenciar determinado objeto de estudo para serem observadas,

(d) *survey*, interrogação direta de um grupo de pessoas cujo comportamento tem interesse, (e) de campo, consiste na observação de fatos e de fenômenos tal como ocorrem espontaneamente, (f) estudo de caso, envolve o estudo profundo e exaustivo de um ou mais objetivos de modo a permitir mais conhecimento, (g) *ex-post-facto,* a pesquisa ocorre após algum fato, (h) pesquisa-ação, realizada para fornecer uma solução para determinado problema, ou (i) pesquisa participante, envolve a interação entre os pesquisadores e os membros das situações investigadas. Este trabalho pode ser classificado como **estudo de caso**, pois foram desenvolvidas e mensuradas três versões de uma LPS com diferentes tecnologias de gerenciamento de variabilidades;

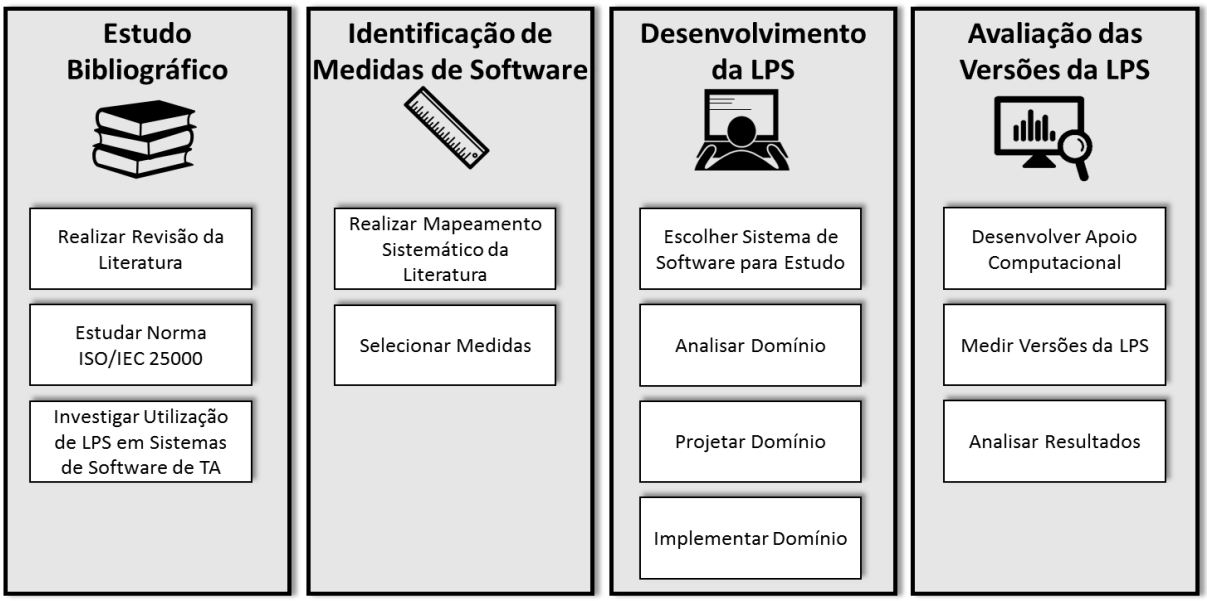
- **Abordagem do problema**. Uma pesquisa pode ser classificada em (a) quantitativa, as opiniões e as informações são traduzidas em números de modo a classificá-las e analisá-las, ou (b) qualitativa, há relação dinâmica entre o mundo real e o sujeito, não sendo traduzível em números. Este trabalho pode ser classificado como uma **pesquisa quantitativa**, pois as versões da LPS são mensuradas utilizando um conjunto de medidas de software e, posteriormente, esses dados são analisados utilizando técnicas estatísticas, por exemplo, teste de normalidade de Kolmogorov-Smirnov, para identificar a distribuição dos dados, o teste de hipóteses de Friedman, para identificar se existe diferença significativa entre as medidas de software para modularidade utilizando diferentes tecnologias de gerenciamento de variabilidades, e o teste *post-hoc* de Conover, para agrupar as tecnologias de gerenciamento de variabilidades conforme a diferença significativa identificada entre elas.

2.3 Método de Pesquisa

O presente trabalho iniciou-se em setembro de 2017 e foi finalizado em julho de 2019. Para sua execução, foi elaborado e seguido o seguinte método de pesquisa com as seguintes etapas (Figura 2.1):

- **Estudo Bibliográfico**. Nessa etapa, o propósito é conhecer o tema de estudo. Para isso, foram contemplados os fundamentos teóricos, bem como saber como LPS é utilizada para o desenvolvimento de sistemas de software de TA. Essa etapa possui três atividades:
  - **Realizar Revisão da Literatura**. Foram realizadas pesquisas *ad hoc* sobre LPS e Tecnologias para Gerenciamento de Variabilidades;

Figura 2.1 - Método de Pesquisa



- **Estudar Norma ISO/IEC 25000**. Na série de normas ISO/IEC 25000:2011 (SQuaRE - *System and Software Quality Requirements and Evaluation*), o modelo de qualidade de sistemas de software é definido pela norma ISO/IEC 25010:2011. Esse modelo contém oito características de qualidade de sistemas de software e suas respectivas subcaracterísticas. A Manutenibilidade é uma característica de qualidade e contém como subcaracterísticas: Analisabilidade, Modularidade, Modificabilidade, Reusabilidade e Testabilidade. Com isso, pode-se aprofundar o entendimento da subcaracterística Modularidade para ser utilizada em LPS;
- **Investigar Utilização de LPS em Sistemas de Software de TA**. Foi realizada uma revisão literatura *ad hoc* sobre a utilização de LPS para desenvolvimento de sistemas de software de TA. Em particular, nessa revisão, o foco foi na forma de realizar e de avaliar a qualidade das atividades da engenharia de LPS. Como resultado, foi identificado que LPS ainda é pouco difundida para o desenvolvimento de sistemas de software de TA e, principalmente, pouco se sabe a respeito da qualidade dos processos empregados para seu desenvolvimento. Assim, para contribuir com o estado atual da literatura, o estudo de caso foi destinado para o desenvolvimento de uma LPS de TA, mostrando os processos de desenvolvimento relacionados a Engenharia de Domínio da LPS (focando no desenvolvimento de ativos da LPS, não na derivação de seus produtos) e para a avaliação da qualidade da LPS em termos de sua modularidade (em especial, coesão e acoplamento de *features*);

- **Identificação de Medidas de Software**. Nessa etapa, o propósito é identificar as medidas de software relacionadas à modularidade, considerando a coesão e o acoplamento de *features* em LPS. Essa etapa possui duas atividades:
  - **Realizar Mapeamento Sistemático da Literatura**. Foi realizado um Mapeamento Sistemático da Literatura (MSL) para identificar as medidas de software frequentemente utilizadas para mensurar a qualidade em LPS. As medidas identificadas foram relacionadas às características de qualidade definidas na norma ISO/IEC 25010:2011, aos processos da engenharia de LPS, às propriedades de software (*e.g.*, tamanho, complexidade, acoplamento e coesão) e às tecnologias de gerenciamento de variabilidades;
  - **Selecionar Medidas**. Consistiu em selecionar as medidas mais presentes nos artigos encontrados no MSL para avaliar a modularidade em LPS, quanto à coesão e ao acoplamento de suas *features*;
- **Desenvolvimento da LPS**. Nessa etapa, o propósito foi o desenvolvimento de uma LPS a partir do sistema de software escolhido para o estudo de caso (*plug-in* `WebHelpDyslexia`), utilizando as três tecnologias de gerenciamento de variabilidades. Essa etapa possui quatro atividades:
  - **Escolher Sistema de Software para Estudo**. A escolha consistiu na busca por sistemas de software de TA. Os seguintes critérios foram utilizados nessa escolha: i) ser um recurso de TA; ii) possuir código aberto; e iii) ser desenvolvido na linguagem de programação Java. Para isso, foram analisados os recursos presentes no repositório *ATHENA: Free AT Software Inventory*[1] e, posteriormente, em outros repositórios. Como não foram encontrados sistemas de software de TA satisfatórios, optou-se por utilizar o *plug-in* `WebHelpDyslexia`, por causa do conhecimento prévio do seu domínio;
  - **Analisar Domínio**. Consistiu na análise de sistemas de software desenvolvidos para auxiliar pessoas com dislexia na leitura e na compreensão de textos Web. Assim, foi possível identificar as *features* comuns e variáveis para esse tipo de sistemas;
  - **Projetar Domínio**. Consistiu na identificação das *features* comuns e variáveis e seus relacionamentos para sua modelagem, utilizando o *plug-in* FeatureIDE[2];

[1] Disponível em: <https://access.uoa.gr/ATHENA/eng/pages/home>
[2] Disponível em: < https://featureide.github.io/>

- **Implementar Domínio**. Originalmente, o *plug-in* `WebHelpDyslexia` foi desenvolvido como extensão do Google Chrome, utilizando a linguagem JavaScript. Dessa forma, foi desenvolvida uma versão *desktop* desse *plug-in*, utilizando a linguagem Java. Posteriormente, as linguagens AspectJ e Jakarta foram utilizadas para o desenvolvimento da LPS nas seguintes tecnologias de gerenciamento de variabilidades: i) Orientação a Aspectos, foi utilizada a linguagem AspectJ; ii) Orientação a Características, foi utilizada a linguagem Jakarta (AHEAD - *Algebraic Hierarchical Equations for Application Design*); e iii) Módulos de Características Aspectuais, foram utilizadas as linguagens AspectJ e Jakarta;

- **Avaliação das Versões da LPS**. Nessa etapa, o propósito é realizar a medição das versões da LPS implementada nas três tecnologias de gerenciamento de variabilidades. Essa etapa possui três atividades:

  - **Desenvolver Apoio Computacional**. Foi desenvolvida uma ferramenta computacional para coletar o valor das medidas nas versões da LPS. Para isso, foi utilizada a linguagem de programação Java e suas bibliotecas para a representação do código em árvore sintática abstrata (*Abstract Syntax Tree* - AST);
  - **Medir Versões da LPS**. Foi coletado o valor de medidas de acoplamento e de coesão nas LPS implementadas nas três tecnologias de gerenciamento de variabilidades. Para a coleta desses valores, foi utilizado o apoio computacional desenvolvido (*plug-in* para IDE Eclipse) na atividade anterior;
  - **Analisar Resultados**. Os dados coletados foram analisados para identificar se existe diferença significativa estatisticamente entre as três tecnologias de gerenciamento utilizadas para o desenvolvimento das LPS. Para isso, foram definidas as hipóteses:

    H_0: A modularidade da `LPS WebHelp` desenvolvida com as tecnologias de Orientação a Características, de Orientação a Aspectos e de Módulos de Características Aspectuais são equivalentes.

    H_1: A modularidade da `LPS WebHelp` desenvolvida com as tecnologias de Orientação a Características, de Orientação a Aspectos e de Módulos de Características Aspectuais não são equivalentes.

    Para refutar as hipóteses, foram utilizados testes estatísticos. Primeiro, foi verificada a normalidade dos dados por meio do teste de Kolmogorov-Smirnov. Em seguida, o teste de Friedman foi selecionado para verificar se existe diferença significativa entre as tecnologias de gerenciamento de variabilidades.

2.4 Considerações Finais

Este trabalho pode ser classificado como: i) uma pesquisa aplicada, pois é investigado o valor de medidas de acoplamento e de coesão software para a modularidade de uma LPS desenvolvida com três tecnologias de gerenciamento de variabilidades; ii) uma pesquisa descritiva, pois objetiva investigar se existe diferença significativa estatisticamente entre a modularidade da LPS desenvolvida com três tecnologias de gerenciamento de variabilidades; iii) um estudo de caso, pois foram desenvolvidas e mensuradas três versões de uma LPS desenvolvidas com diferentes tecnologias de gerenciamento de variabilidades; e iv) uma pesquisa quantitativa, pois as versões da LPS são mensuradas utilizando um conjunto de medidas de software e, posteriormente, esses dados são analisados utilizando técnicas estatísticas.

O trabalho seguiu o método de pesquisa proposto, composto por quatro etapas: i) Estudo Bibliográfico; ii) Identificação de Medidas de Software; iii) Desenvolvimento da LPS; e iv) Avaliação das Versões da LPS. Cada etapa é constituída por atividades, totalizando doze atividades necessárias para alcançar o objetivo desta pesquisa.

3 LINHAS DE PRODUTOS DE SOFTWARE

3.1 Considerações Iniciais

A demanda por sistemas de software cada vez maiores e mais complexos torna essencial o rápido desenvolvimento desses sistemas que apresentem mais qualidade e preço mais acessível (POHL et al., 2005). De modo a atender essa demanda, LPS pode ser empregada, uma vez que viabiliza a produção em massa de produtos customizados por meio do reúso de software (APEL et al., 2013).

O restante deste capítulo está organizado da seguinte forma. Na Seção 3.2, são introduzidos conceitos importantes sobre LPS. Na Seção 3.3, é discutido o processo de desenvolvimento de uma LPS, que engloba os processos de Engenharia de Domínio e de Engenharia de Aplicação. Na Seção 3.4, são descritas as abordagens Proativa, Reativa e Extrativa utilizadas para a geração/obtenção de uma LPS. Na Seção 3.5, é apresentada a representação comumente utilizada para modelar LPS: Modelo de Características (*Feature Model - FM*). Na Seção 3.6, são apresentadas as tecnologias de gerenciamento de variabilidades utilizadas para o desenvolvimento de LPS. Na Seção 3.7, são mostradas vantagens e desvantagens na utilização de LPS.

3.2 Conceitos

Uma LPS é formada por um conjunto de produtos (sistemas) de software estritamente relacionados e desenvolvidos sob uma base de código comum (THÜM et al., 2014a). Os produtos de software gerados a partir de uma LPS diferem-se em relação aos seus recursos fornecidos. A ideia de LPS é construir produtos de software personalizados de forma automática por meio da seleção de um conjunto de recursos (APEL et al., 2013). Esses recursos, também conhecidos como características (*features*), são um conjunto de artefatos de software reutilizáveis, que englobam, por exemplo, artefatos de código, modelos de requisitos e de arquitetura, componentes de software e planos de teste.

Esses artefatos devem ser reutilizados de forma consistente e sistemática para construir aplicativos robustos (SILVA et al., 2011). Por conseguinte, LPS propicia redução do tempo de mercado, melhoria da qualidade de produtos de software (THÜM et al., 2014a) e desenvolvimento de produtos sob medida para clientes individuais (APEL et al., 2013). Ao fornecer um produto sob medida por meio da combinação de recursos reutilizáveis, o custo é

reduzido por não ter que desenvolver o produto do “zero” (desde o início). Além disso, o reúso de recursos gera aumento na qualidade dos produtos, uma vez que os recursos foram sistematicamente testados em muitos produtos. Por fim, o produto pode ser produzido rapidamente por meio da combinação e da adaptação de recursos existentes. Assim, as diferenças entre LPS e a abordagem tradicional de desenvolvimento de sistemas de software tornam-se latentes, por exemplo (KANG et al., 2009):

- LPS é destinada ao desenvolvimento de projetos que contemplam uma linha de produtos (sistemas), ao passo que a abordagem tradicional está relacionada ao desenvolvimento de projetos individuais;
- Com a abordagem tradicional, pode-se realizar a produção em larga escala de produtos ao custo da diminuição da sua personalização, enquanto, com LPS, há produção em larga escala de produtos, permitindo aumento na personalização desses produtos.

3.3 Processo de Desenvolvimento de Linhas de Produtos de Software

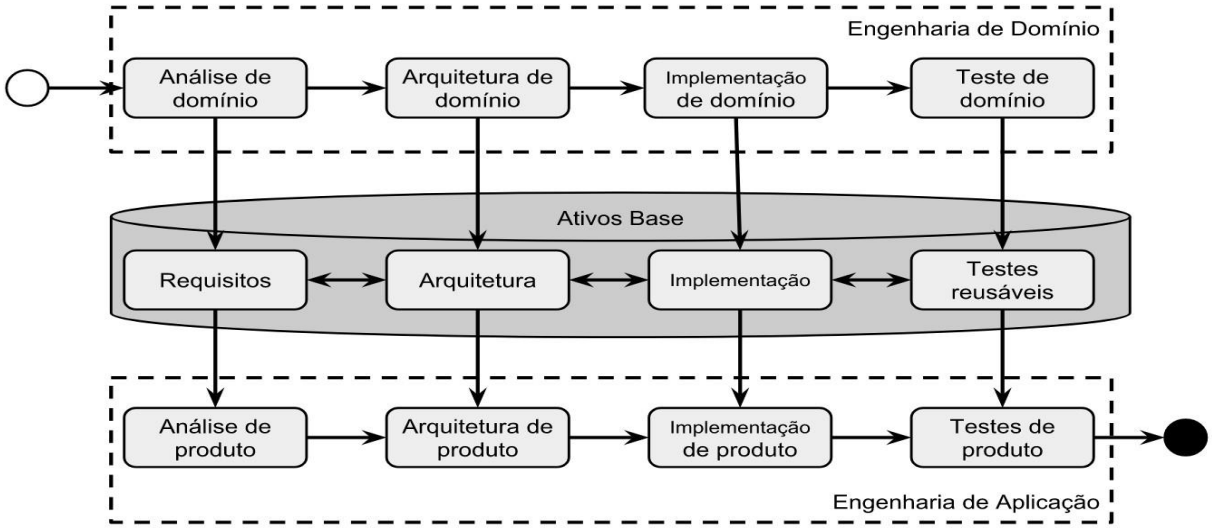
No desenvolvimento de uma LPS, há os processos Engenharia de Domínio e Engenharia de Aplicação, cujo ciclo de vida pode ser acompanhado na Figura 3.1. Em ambos os ciclos, são realizadas atividades de análise, de arquitetura, de implementação e de testes (APEL et al., 2013; SILVA et al., 2011):

- **Engenharia de Domínio** é o processo no qual a uniformidade e a variabilidade de LPS são definidas. Esse processo não resulta em um produto de software específico, mas prepara os artefatos para serem utilizados em diversos produtos. As atividades são:
  - **Análise de Domínio**. São contempladas as atividades para entendimento do domínio. Para tanto, é necessário identificar os requisitos comuns e os requisitos variáveis relevantes. Grande quantidade de requisitos comuns em uma LPS, geralmente, requer menor esforço para projetar e realizar a LPS. Em contrapartida, com a extensão dos requisitos variáveis, há determinação da quantidade potencial de diferentes aplicações que podem ser derivadas da LPS o que implica na quantidade de segmentos de mercado que podem ser satisfeitos;
  - **Arquitetura de Domínio**. Há abrangência das atividades para definir a arquitetura de referência da LPS, sendo definidos os pontos de variação e um conjunto de variantes para a LPS. A fim de implementar uma arquitetura de referência, são utilizados métodos e técnicas para a modelagem das variabilidades. Geralmente,

essa modelagem é feita por meio de um grafo direcionado acíclico ou árvore que representa as *features* da LPS e seus relacionamentos;

- **Implementação de Domínio**. São consideradas as atividades de implementação dos artefatos reutilizáveis. Nessa atividade, são utilizados os recursos das linguagens de programação existentes, compiladores e *linkers*. Geralmente, para a implementação dos artefatos, são utilizadas as tecnologias de Orientação a Aspectos, Orientação a Características e Orientação a Delta;
- **Teste de Domínio**. São reveladas as evidências de defeitos nos artefatos antes de serem reutilizados no processo de Engenharia de Aplicação;

Figura 3.1 - Engenharia de Domínio e Engenharia de Aplicação (Adaptado de Silva et al. (2011))



- **Engenharia de Aplicação** é processo no qual são reutilizados os artefatos desenvolvidos na Engenharia de Domínio para a construção de sistemas. Nesse processo, a variabilidade e a comunalidade da LPS são exploradas de modo a reutilizar os artefatos de domínio na construção de produtos específicos. As atividades são:

  - **Análise de Produto**. Os requisitos para uma aplicação específica são determinados. Para tanto, é definida uma relação entre os requisitos de domínio e os recursos da aplicação correspondente;
  - **Arquitetura de Produto**. As variabilidades identificadas na Arquitetura de Domínio são avaliadas em relação aos requisitos da aplicação. Dessa forma, são escolhidas as variantes que se encaixam da melhor forma;
  - **Implementação do Produto**. Os artefatos de código são ajustados com base na Arquitetura de Produto e na Análise de Produto. As técnicas de configuração de

software podem ser utilizadas para gerar automaticamente os produtos da LPS, uma vez que essas técnicas permitem a composição dos módulos de código reutilizáveis;

- **Teste de Produto**. O objetivo é testar, de forma abrangente, os produtos da LPS. Assim, é verificada e validada uma aplicação em relação a sua especificação, garantindo a corretude das variantes e seus efeitos colaterais.

3.4 Abordagens de Construção

Existem três abordagens principais para a construção de uma LPS (SILVA et al., 2011):

- **Abordagem Proativa**. No desenvolvimento dos artefatos da LPS, o ponto de partida é a análise, o projeto e a implementação de domínio e, posteriormente, a Engenharia de Aplicação. Nessa abordagem, o objetivo é projetar uma LPS com base nos possíveis sistemas a serem desenvolvidos, o que requer investimento antecipado nos ativos de produção. Dessa forma, quando os requisitos para a criação de um conjunto de produtos estão estáveis, essa abordagem é a mais apropriada (Figura 3.2);
- **Abordagem Extrativa**. O desenvolvimento da LPS parte de um conjunto de sistemas de software existentes, de modo que possam ser criados sistemas com finalidades semelhantes. Para tanto, *features* comuns e *features* variáveis dos sistemas devem ser extraídas para serem reutilizadas (Figura 3.3);
- **Abordagem Reativa**. A LPS é desenvolvida incrementalmente e é ampliada quando há demanda de novos requisitos ou produtos (Figura 3.4).

Figura 3.2 - Abordagem Proativa

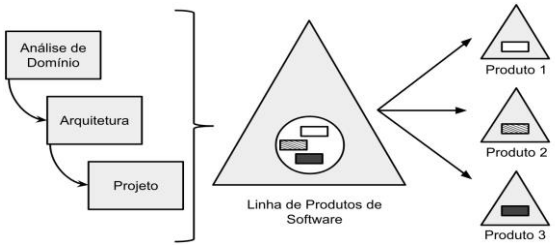


Figura 3.3 - Abordagem Extrativa

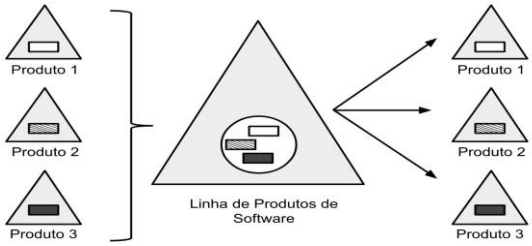
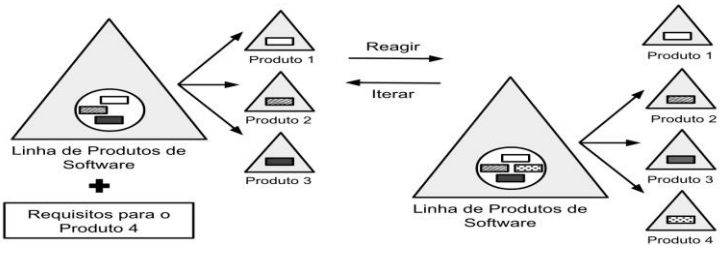


Figura 3.4 - Abordagem Reativa



3.5 Modelagem de Características

As comunalidades e as variabilidades entre produtos de uma LPS podem ser expressas por *features*. Originalmente, o conceito de *feature* foi introduzido com a modelagem *Feature Oriented Domain Analysis* (FODA), sendo um diagrama em forma de árvore para representar as *features* e suas associações, que podem ser do tipo *Mandatory*, *Optional* e *Alternative-constraint* (KANG et al., 1990). A partir dessa modelagem, diversas extensões foram propostas, por exemplo *Feature Reuse-Driven Software Engineering Business* (*Feature*-RSEB) que apresenta a associação *Unrestricted-or* (GRISS et al., 1998); a extensão para adição de informações sobre *binding-time* (tempo em que a *feature* deve ser selecionada: compilação ou execução) (GURP et al., 2001); e a extensão para adição de cardinalidade nos relacionamentos entre *features* (CZARNECKI et al., 2005). Em geral, as associações entre as *features* podem ser classificadas em (APEL et al., 2013):

- ***Mandatory feature***. Corresponde a uma equivalência lógica, na qual deve-se selecionar a *feature* filho `f` ao selecionar a *feature* pai `p` e vice-versa. A equivalência é denotada por um círculo preenchido em `f` (Figura 3.5) e definida como

$$mandatory(p, f) \equiv f \leftrightarrow p$$

- ***Optional feature***. Corresponde a uma implicação lógica, na qual deve-se selecionar a *feature* pai `p` correspondente ao selecionar a *feature* filho `f`. A implicação é denotada por um círculo vazio em `f` (Figura 3.6) e definida como

$$optional(p, f) \equiv f \Rightarrow p$$

- ***Unrestrict Or***. Corresponde a uma disjunção lógica, na qual podem ser escolhidos uma ou mais *features* filho de um conjunto não vazio $f_1, \ldots, f_n$ para uma *feature* pai `p`. A disjunção é denotada por um semicírculo preenchido em `p` (Figura 3.7) e definida como

$$or(p, \{f_1, \ldots, f_n\} \equiv ((f_1 \vee \ldots \vee f_n) \Leftrightarrow p$$

- ***Alternative constraint***. Corresponde a uma disjunção exclusiva, na qual deve ser escolhida exatamente uma *feature* filho de um conjunto não vazio $f_1, \ldots, f_n$ para uma *feature* pai `p`. A disjunção exclusiva é denotada por um semicírculo vazio em `p` (Figura 3.8) e definida como

$$alternative(p, \{f_1, \ldots, f_n\}) \equiv \left((f_1 \vee \ldots \vee f_n) \Leftrightarrow p\right) \bigwedge_{i<j} \neg\, (f_i \wedge f_j)$$

Figura 3.5 - *Mandatory Feature*

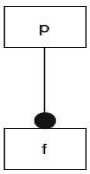


Figura 3.6 - *Optional Feature*

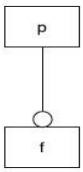


Figura 3.7 - *Unrestricted Or*

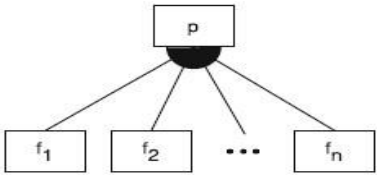
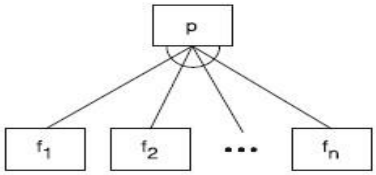
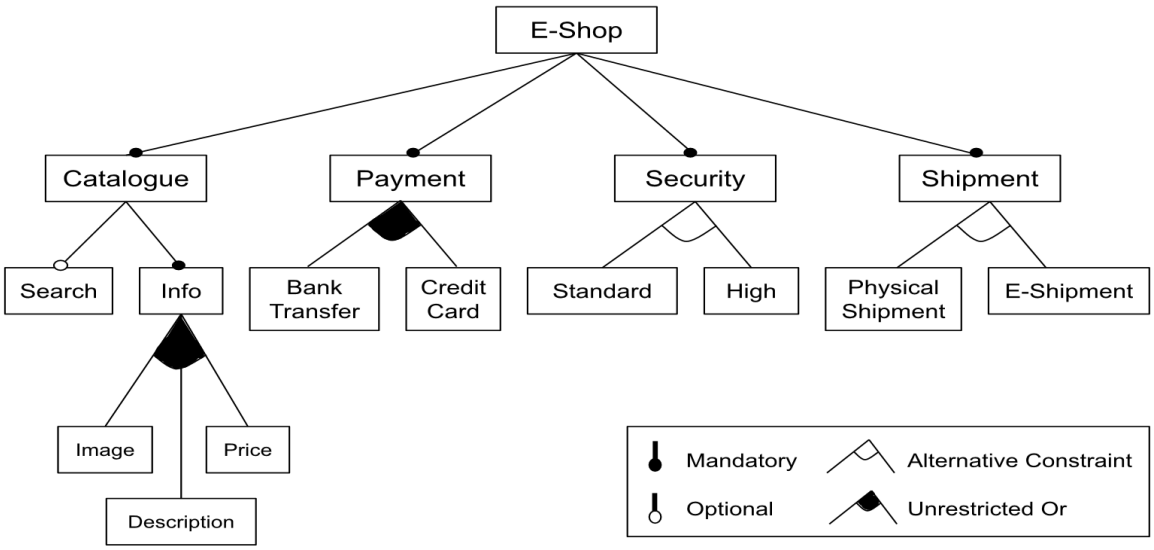


Figura 3.8 - *Alternative Constraint*



Como exemplo, um domínio de sistema de `E-Shop` é apresentado na Figura 3.9. Cada produto de software pertencente a LPS deve fornecer um catálogo de produtos (`Catalogue`), uma forma de pagamento (`Payment`), segurança (`Security`) para operações realizadas e uma forma de envio (`Shipment`). O catálogo fornece, opcionalmente, um mecanismo para facilitar a busca de produtos (`Search`) e fornece informações (`Info`) obrigatórias sobre os produtos. O pagamento pode ser feito por transferência bancária (`Bank Transfer`) e/ou cartão de crédito (`Credit Card`), ambas podem ser disponibilizadas no sistema. A segurança dos produtos gerados pode ser padrão (`Standard`) ou de nível alto (`High`), sendo que deve ser selecionada exatamente uma das opções. O tipo de envio pode ser físico (`Physical Shipment`) ou eletrônico (`E-Shipment`), sendo que deve ser selecionada exatamente uma das opções. Os produtos presentes no catálogo podem conter uma ou mais informações, como imagem (`Image`), descrição (`Description`) e preço (`Price`).

Figura 3.9 - Exemplo de Modelo de Variabilidade (Adaptado de Segura et al. (2010))



Portanto, o modelo de características (*features*) é um documento de referência, sendo utilizado com o propósito de fornecer uma base para a infraestrutura destinada a reutilização e provê uma visão geral dos produtos de uma LPS.

3.6 Tecnologias para Gerenciar Variabilidades

O modelo de características descreve uma LPS em termos dos diferentes recursos do sistema e das restrições existentes (SILVA et al., 2011). A partir desse modelo, podem ser empregados configuradores que suportam a propagação de escolhas e a auto conclusão para a derivação dos produtos de uma LPS (ARCAINI et al., 2017). Para a geração automática de sistemas de software, existem métodos e ferramentas que proporcionam o desenvolvimento de uma LPS a partir de especificações em alto nível. Para isso, é utilizado o Desenvolvimento Generativo, uma técnica que consiste no mapeamento de requisitos em configuração, separando em (CZARNECKI; EISENECKER, 2000 apud JAVED et al., 2016): i) espaço problema, contém conceitos e características de um domínio; e ii) espaço solução, contém componentes de implementação que visam maximizar a reutilização.

Após essa separação, é feito o mapeamento entre o espaço problema e o espaço solução. Esse mapeamento pode ser considerado sob a perspectiva de visão de configuração, na qual o espaço problema é um conjunto de características a serem selecionadas e o espaço solução consiste em um conjunto de componentes a serem combinados (CIRILO, 2008). Além disso, o mapeamento pode ser visto sob a perspectiva da visão transformacional, na qual o espaço problema é representado por uma linguagem específica de domínio e o espaço solução por uma linguagem de programação. De modo a auxiliar no mapeamento entre espaço problema e espaço solução, nesta seção, são abordadas as tecnologias utilizadas para o gerenciamento de variabilidades em LPS.

Esta seção está organizada da seguinte forma. Na Seção 3.6.1, considerando a visão de configuração, são apresentadas as tecnologias baseadas em anotação que podem ser categorizadas em anotação visual e em anotação textual. Na Seção 3.6.2, considerando a visão transformacional, são apresentadas as tecnologias baseadas em composição, por exemplo, Orientação a Aspectos, Orientação a Características e Módulos de Características Aspectuais.

3.6.1 Tecnologias Baseadas em Anotação

Nas tecnologias baseadas em anotação, é sugerido que o código das *features* da LPS permaneça entrelaçado ao código base do sistema. Isso significa que o espaço solução está

espalhado entre diferentes artefatos, de modo contrário a definição do espaço problema, pois ele é definido por um artefato de *design* (ARCAINI *et al*., 2017). Dessa forma, cabe construir um modelo para o espaço solução utilizando anotações explícitas.

Com as anotações explícitas, são acrescentadas metainformações diretamente no código para delimitar o código a ser analisado por um pré-processador (GAIA, 2013). Com isso, o código é examinado por um sistema de software antes da fase de compilação para as modificações serem executadas e o código ser repassado ao compilador. Assim sendo, o modelo para o espaço solução pode ser construído utilizando anotações explícitas textuais ou visuais.

3.6.1.1 Anotação Textual

Com as anotações textuais, é indicado por meio de diretivas de pré-processamento, o código pertencente a determinada *feature*, uma vez que as *features* estão localizadas em um mesmo trecho de código. O código de uma *feature* pode estar entrelaçado com o código de outras *features*. Uma forma de realizar a anotação textual é com a Compilação Condicional, na qual são utilizadas diretivas `#ifdef`, como as utilizadas pelo pré-processador das linguagens C/C++, para realizar as marcações no código (MONTALVILLO; DÍAZ, 2016). Assim, as *features* não marcadas têm seu código removido em tempo de compilação para gerar o produto.

No algoritmo mostrado na Código 3.1, é apresentado um exemplo de marcação de código utilizando diretivas C/C++. Na linha 2, foi definido, por meio da diretiva `#define`, um identificador nomeado `ID`. Esse identificador é passado para a diretiva `#if` (linha 6), na qual é feita a avaliação da expressão. Caso a expressão seja verdadeira, será impressa a frase "`ID is defined`" (linha 7). Na linha 3, há a diretiva `#undef` que remove a definição atual do identificador. Ao remover o símbolo de comentário (`//`) na linha 3, a definição do `ID` é removida. Dessa forma, o identificador analisado na diretiva `#if` (linha 6) não foi definido, ou seja, a diretiva `#if` se torna falsa. Portanto, a instrução seguinte, expressa pela diretiva `#else` se torna verdadeira (linha 8), para qual é impressa a frase "`ID is not defined`" (linha 9).

A vantagem de utilizar a Compilação Condicional consiste na marcação de código em diferentes níveis de granularidade; no caso de pré-processadores C/C++, as construções da linguagem são isoladas em nível de linhas. No entanto, sua utilização possui desvantagens. Por exemplo, durante a utilização dessa tecnologia, o espaço problema pode divergir do espaço solução (ARCAINI et al., 2017). Isso permite que o espaço solução tenha configurações não permitidas no espaço problema ou pode permitir que o espaço problema tenha configurações não compiláveis. Essa relação pode ser observada na Figura 3.10, em que `P` indica o espaço

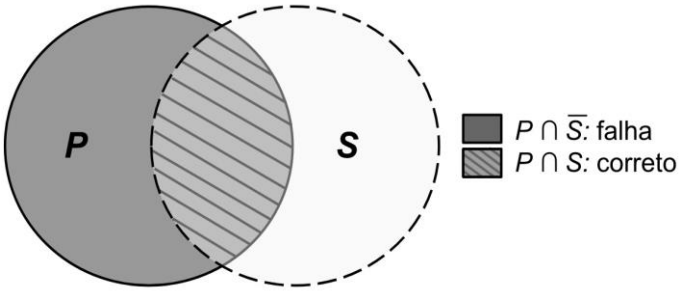
problema e S indica o espaço solução. Apesar de P e S não serem sempre iguais, é necessário que as configurações de P sejam permitidas por S. Desse modo, os produtos corretamente instanciados são representados pela intersecção entre P e S. Os produtos que representam falhas são indicados pelas configurações em P, mas não presentes em S.

Código 3.1 - Uso de Diretivas do Pré-Processador C/C++

```
 1 #include <stdio.h>
 2 #define ID
 3 // #undef ID
 4
 5 int main(){
 6    #if defined(ID)
 7        printf("ID is defined");
 8    #else
 9        printf("ID is not defined");
10    #endif
11    return 0;
12 }
```

Ainda, nessa tecnologia, o código não é extraído fisicamente, requerendo suporte adicional para manter e entender código variável (FERREIRA et al., 2016; QUEIROZ et al., 2017). Isso deve-se à diminuição da legibilidade do código por meio da inserção das diretivas e aos modelos complexos de dependência de código configurável.

Figura 3.10 - Produtos Falhos e Corretos (ARCAINI et al., 2017)

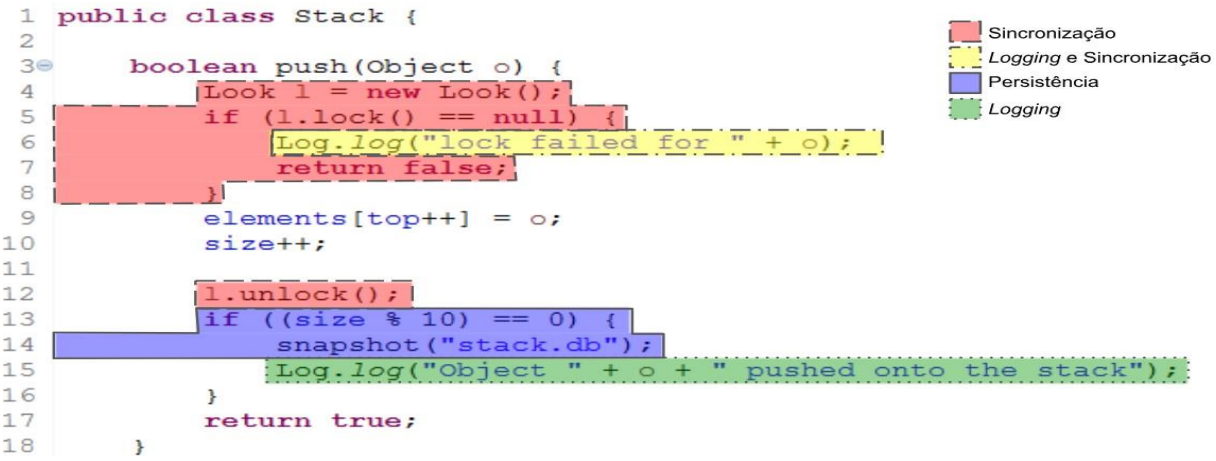


3.6.1.2 Anotação Visual

As anotações não precisam ser apenas declarações textuais de pré-processador. Elas podem ser feitas de forma visual, por meio da utilização de ferramentas como a CIDE (*Colored Integrated Development Environment*) (KÄSTNER; APEL, 2008). Com as ferramentas de anotações visuais, pode-se marcar código e outros artefatos de software. Na ferramenta CIDE, são utilizadas cores de fundo diferentes para anotar trechos de código que implementam *features* distintas. Conforme a documentação disponível (KÄSTNER et al., 2010), com essa ferramenta, o espaço problema converge com o espaço solução, diferente da anotação textual.

Na Figura 3.11, é mostrada a anotação visual utilizando a coloração de código, permitindo melhoria na legibilidade, pois não há diretivas de pré-processamento inseridas no código. Além disso, com a anotação visual, a marcação de código pode ser feita em diferentes níveis de granularidade, visto que as construções da linguagem podem ser isoladas em nível de linhas. Por exemplo, na Figura 3.11, é mostrada a implementação da classe `Stack`. As linhas de código pertencentes à *feature* `Sincronização` (linhas 4 - 8 e 12) estão anotadas com uma cor (no caso, cor vermelha), as linhas de código pertencentes à *feature* `Logging` (linhas 6 e 15) estão anotadas com outra cor (no caso, cor verde) e as linhas de código pertencentes à *feature* `Persistência` (linhas 13 e 14) estão anotadas com outra cor (no caso, cor azul). Como a linha 6 é comum para as *features* `Logging` e `Sincronização`, é anotada com a cor amarela, correspondente a composição das cores vermelha e verde (COUTO, 2010).

Figura 3.11 - Ferramenta CIDE (Adaptado de Couto et al. (2010))



Apesar de melhorar a visibilidade do código, as anotações visuais são feitas de forma manual, o que pode ocasionar erros, principalmente no desenvolvimento de LPS que não são do domínio de conhecimento dos desenvolvedores (COUTO, 2010). Outro problema relaciona-se a utilização de cores para a marcação de *features*, pois há uma quantidade limitada de cores, o que pode dificultar a marcação e a identificação das *features* da LPS.

3.6.2 Tecnologias Baseadas em Composição

Nas tecnologias baseadas em composição, o objetivo é a separação física entre o código base e o código da *feature*, sendo cada *feature* implementada em um módulo distinto. Para isso, as tecnologias baseadas em composição fornecem mecanismos para definir pontos de extensão, que permitem a composição do código em *features* (KÄSTNER et al., 2008). Dessa forma, esses pontos de extensão limitam as tecnologias baseadas em composição a uma granularidade

grossa. Isto é, as construções da linguagem não são separadas em nível de linhas, mas em relação aos mecanismos de decomposições estruturais oferecidos pela linguagem; por exemplo, as unidades de código representadas por classes em Orientação a Objetos. Assim, nesta seção, são abordadas as tecnologias baseadas em composição que são objetos de investigação nessa pesquisa: i) Orientação a Aspectos; ii) Orientação a Características; e iii) Módulos de Características Aspectuais.

3.6.2.1 Orientação a Aspectos

Com Orientação a Aspectos (OA), é possível a separação e a organização do código de acordo com a sua relevância para a aplicação (GIL et al., 2015). Um dos elementos centrais para OA é o conceito de interesses, que representam características relevantes para a aplicação. Muitas vezes, pode ocorrer o entrelaçamento de interesses no código (*tangled code*) ou o espalhamento de um interesse por diversos módulos do sistema de software (*scattering code*). Tais interesses são conhecidos como interesses transversais. A fim de modularizar os interesses transversais, OA é utilizada, visto que esses interesses não podem ser adequadamente implementados utilizando Orientação a Objetos (OO) (THÜM et al., 2014b).

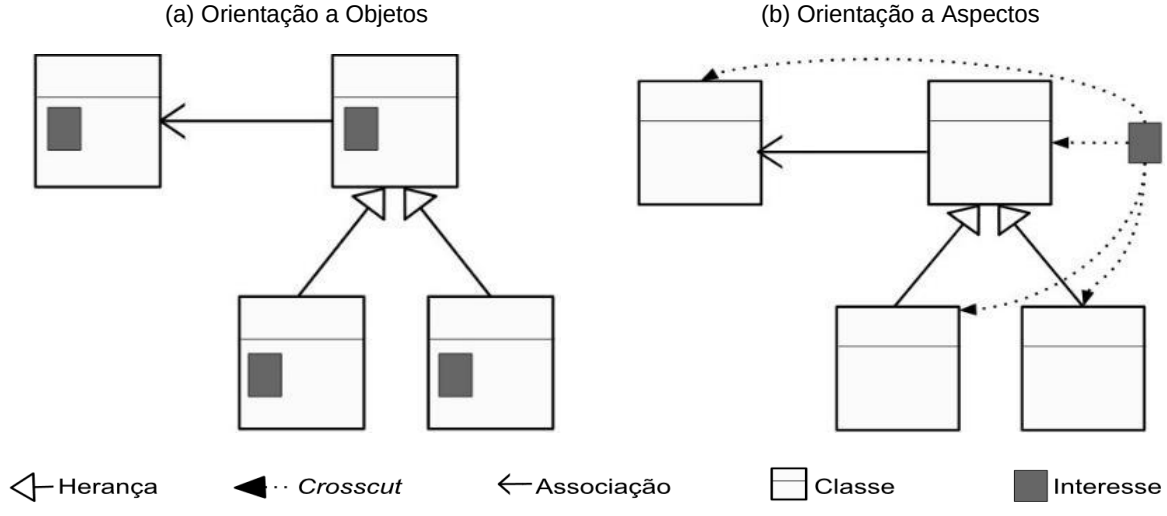
Interesses transversais como `segurança`, `persistência`, `tratamento de exceções`, `otimização de desempenho`, `concorrência` e `tracing` são exemplos de interesses transversais. Em sistemas de software orientados a objetos, esses interesses estão espalhados pelas classes do sistema de software de forma "indisciplinada", o que dificulta a reutilização e a manutenção do código (ASPECTJ TEAM, 2008). Por exemplo, o interesse transversal `tracing`, cujo objetivo é identificar os métodos acessados durante a execução de um sistema de software, pode ser implementado de forma simples em OO, por meio da inserção de linhas de código que mostram mensagens ao executar determinado método. Essa mensagem seria duplicada em diversos métodos para os quais se deseja obter alguma informação, o que ocasionaria espalhamento do código referente ao interesse transversal `tracing`. Com OA, esse interesse pode ser encapsulado em um aspecto que realizaria chamadas dos métodos para os quais se para deseja obter informações.

De modo a exemplificar o espalhamento de código para o interesse transversal `tracing` em OO, foram utilizados quadrados cinza-escuro na Figura 3.12(a), para representar a mensagem inserida nos métodos da classe. Para representar o encapsulamento desse interesse utilizando OA, foi utilizado apenas um quadrado cinza-escuro na Figura 3.12(b). Pode-se notar

que, em OA, há melhor modularização do interesse, diminuindo a redundância e o acoplamento (ASPECTJ TEAM, 2008). Os principais conceitos utilizados em OA são (GIL et al., 2015):

- **Pontos de junção (*Join Points*)**. São posições específicas no fluxo de controle de um sistema de software orientado a objetos, como chamadas de métodos e de construtores, execução de métodos e de construtores e instanciação de objetos;
- **Pontos de corte (*Pointcuts*)**. Referem-se à especificação de múltiplos pontos de junção. Dessa forma, um ponto de corte é uma construção que agrupa um conjunto de pontos de junção e detecta os pontos de junção que devem ser interceptados pelos aspectos;
- **Adendos (*Advices*)**. São estruturas que denotam o comportamento de um aspecto. Há três tipos de adendos:
  - ***Before***. Refere-se à execução dos adendos antes do ponto de junção;
  - ***After***. Refere-se à execução dos adendos após o ponto de junção. O adendo pode ser interceptado quando o método retorna normalmente sua execução (*After returning*) ou quando o método retorna uma exceção (*After throwing*);
  - ***Around***. Os adendos são executados com um comportamento personalizado que pode vir antes ou depois da invocação do método. Além disso, são responsáveis por decidir se o ponto de junção será executado ou não;
- **Declarações de intertipo (*Intertype declarations*)**. São mecanismos que permitem que um aspecto adicione outras declarações em um objeto existente;
- **Aspectos (*Aspects*)**. É a modularização de interesses transversais.

Figura 3.12 - Exemplo do Interesse Transversal `Tracing` em Orientação a Objetos e em Orientação a Aspectos (Adaptado de Apel et al. (2008))



Na OA, a linguagem de programação AspectJ pode ser utilizada por ser considerada mais madura e estável, sendo uma extensão para linguagem Java (THÜM et al., 2014b). No Código 3.2, é apresentado o encapsulamento do interesse transversal `tracing` no aspecto `Tracing.aj`, desenvolvido na linguagem AspectJ. O aspecto é definido na linha 1, para o qual é executado o adendo `before` (linhas 2 e 3) antes da execução de métodos `void *.print()`. Interesses transversais (*e.g.*, `tracing`), são resolvidos com a utilização de extensões homogêneas, ou seja, uma única extensão é aplicada a múltiplos pontos de junção. A implementação de interesses que demandam extensões homogêneas é bem suportada pela OA, por permitir a chamada de vários pontos de junção em um único aspecto.

Código 3.2 - Implementação do Aspecto `Tracing.aj`

```
1 aspect Tracing{
2     before(): execution(void *.print()){
3         System.out.println("Imprime " + variavel);
4     }
5 }
```

3.6.2.2 Orientação a Características

Com Orientação a Características (OC), é possível a modularização de sistemas de software em que as características (*features*) são consideradas as principais abstrações. Além disso, há coexistência de decomposições estruturais, representadas por unidades de código e por unidades de características (SOBERNIG et al., 2016). Normalmente, essa decomposição consiste em: i) unidades de código, são estruturadas de acordo com os mecanismos de decomposição oferecidos por uma linguagem de programação; e ii) unidades de características, originam uma decomposição ortogonal ao objeto orientado, uma vez que incorporam as implementações de recursos na base de código.

Para isso, pode ser utilizado um modelo algébrico de síntese de programas baseado em refinamento gradual, por exemplo, o AHEAD. Na Figura 3.13, é representado o conceito de refinamento gradual, no qual as variantes do sistema de software são obtidas por meio de refinamentos a partir do sistema de software inicial. Nessa figura, cada aresta representa uma característica. Dessa forma, um produto é definido a partir da seleção de um conjunto de características (BENDUHN, 2012).

A decomposição estrutural de um sistema de software consiste em encapsular os fragmentos das classes de um sistema orientado a objetos em *features*. Na Figura 3.14(a), cada característica do software é representada por meio das extensões de classe. Na Figura 3.14(b),

é representado o refinamento gradual, em que uma ou mais classes são encapsuladas em módulos de características (retângulos cinza-claro) de modo a implementar as *features*.

Figura 3.13 - Refinamento Gradual (BENDUHN, 2012)

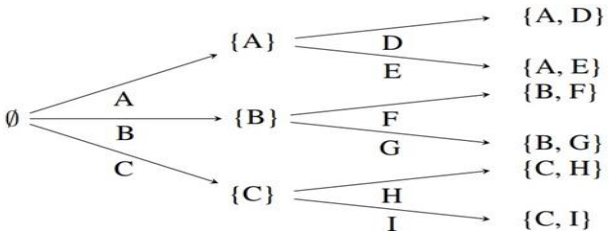
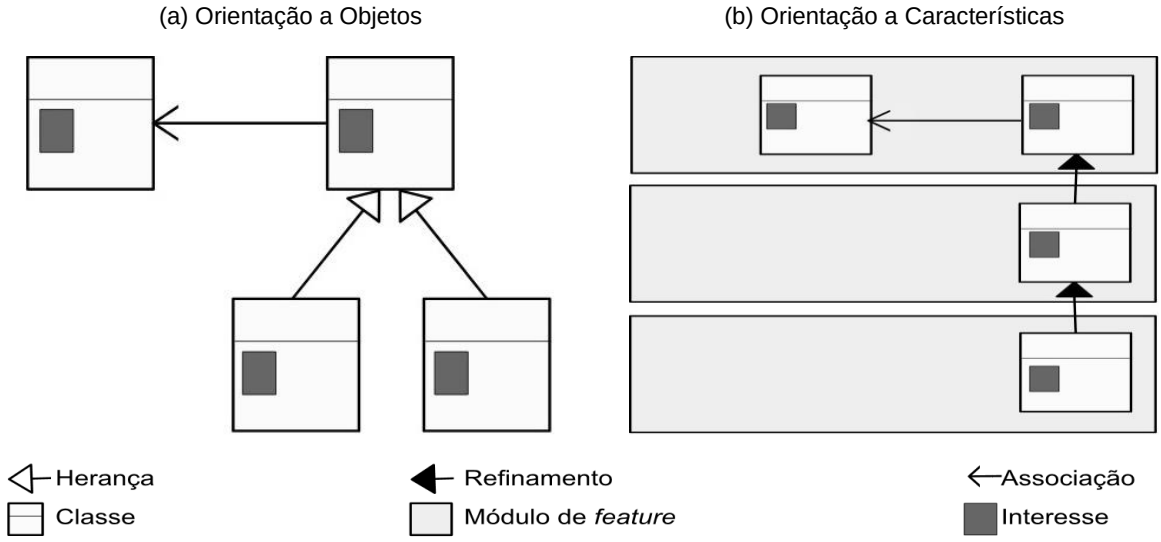


Figura 3.14 - Exemplo do Interesse Transversal `Tracing` em Orientação a Objetos e em Orientação a Características (Adaptado de Apel et al. (2008))



Por exemplo, o interesse transvesal `tracing` poderia ser modularizado em uma *feature* denominada `Tracing`. Com a *suite* AHEAD, as implementações referentes as *features* são encapsuladas em *layers* (diretórios). Assim, no Código 3.3, é apresentada a implementação da classe `Tracing.jak`, desenvolvida na linguagem Jakarta (AHEAD) e encapsulada no diretório referente à *feature* `Tracing`. Nessa classe, há o método sem implementação `public void print()` (linha 2) a ser refinado pelas demais *features* da LPS.

Código 3.3 - Classe `Tracing.jak` pertencente a *Feature* `Tracing`

```
1 public class Tracing {
2     public void print() {}
3 }
```

No Código 3.4, é apresentado o refinamento da classe `Tracing.jak` no diretório de uma *feature* qualquer da LPS, em que o método `public void print()` é refinado (linhas

2 - 5) para mostrar a mensagem contendo as informações sobre a *feature*. Com essa tecnologia, pode-se melhorar a estrutura do sistema, pois encapsula fragmentos de classes relacionados a um interesse em um módulo de *feature*, reduzindo o acoplamento e aumentando a coesão (THÜM et al., 2014b). No entanto, para a implementação do interesse transversal `tracing` em uma *feature*, é necessário aplicar uma única extensão em diversas *features*, não colaborando para a eliminação de código replicado. Assim, OC é melhor explorada em casos em que cada extensão é aplicada em um ponto de junção específico (extensão heterogênea). Por esse motivo, tem-se que OA e OC não são concorrentes, mas complementares (APEL et al., 2008).

Código 3.4 - Refinamento da Classe `Tracing.jak` pertencente a outra *Feature* da LPS

```
1 public refines class Tracing {
2     public void print() {
3         Super.print();
4         System.out.println("Imprime: " + variavel);
5     }
6 }
```

3.6.2.3 Módulos de Características Aspectuais

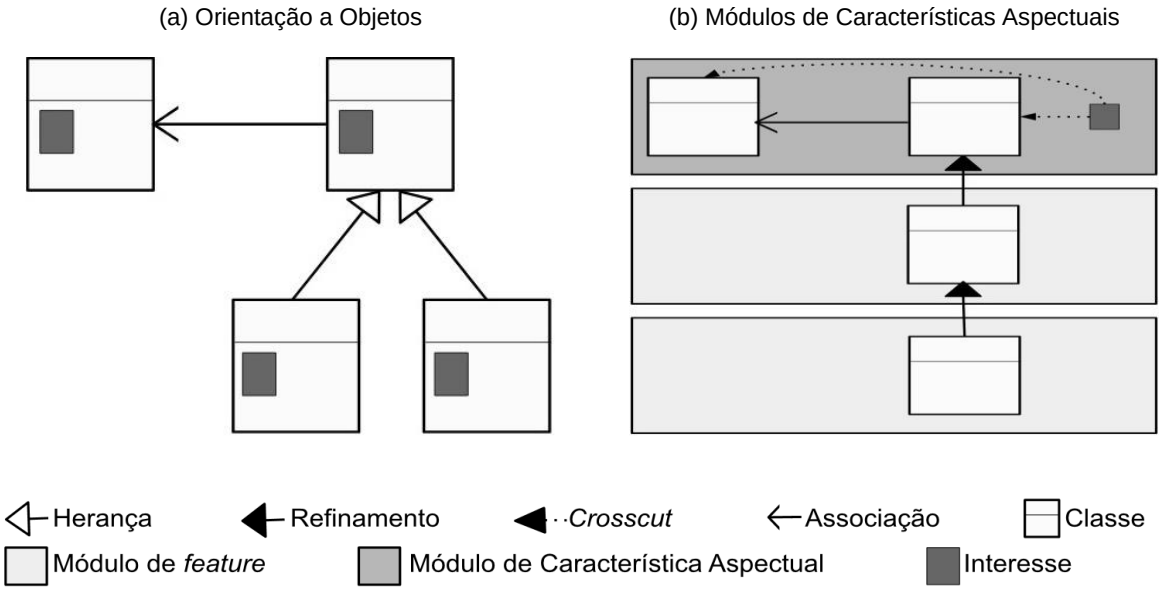
Em Módulos de Características Aspectuais (MCA), há integração de módulos e de aspectos para relacionar OA e OC. Com OC, há a modularização do sistema de software em termos de características, enquanto, com OA, a modularização dos interesses transversais é feita em aspectos. Dessa forma, com Módulos de Características Apectuais, os interesses são modularizados em módulos de *features* e os interesses transversais são modularizados em aspectos e atribuídos aos módulos de *features* correspondentes (GAIA, 2013).

Com MCA, a decomposição de um sistema de software pode ser feita em módulos de características e em módulos de características aspectuais. Na Figura 3.15(a), é apresentado o interesse transversal `tracing` em OO, em que se tem a extensão de classe e o trecho de código referente a esse interesse replicado em cada classe. Na Figura 3.15(b), é apresentada a decomposição do sistema, em que os módulos de características são refinamentos de classes (retângulos cinza-claro) e os módulos de características aspectuais encapsulam aspectos e classes refinadas (retângulo cinza-escuro).

Um sistema de software pode ser decomposto em três dimensões diferentes (APEL et al., 2013): i) Classes e Interfaces; ii) Características; e iii) Aspectos. Com isso, as tecnologias de programação podem ser combinadas a fim de maximizar os benefícios oferecidos por elas. Normalmente, em MCA, um sistema de software é refinado utilizando linguagens que implementam refinamentos (*e.g.* Jakarta) ou mecanismos orientados a aspectos (*e.g.* AspectJ).

Dessa forma, cabe a escolha do tipo de tecnologia que melhor atende ao problema, podendo ser aplicadas separadamente ou em conjunto (APEL et al., 2008).

Figura 3.15 - Exemplo do Interesse Transversal `Tracing` em Orientação a Objetos e em Módulos de Características Aspectuais (Adaptado de Apel et al. (2008))



Por exemplo, o interesse transversal `tracing` é encapsulado em um módulo de *feature*, assim como as demais *features* da LPS. No entanto, como esse interesse é resolvido por meio de uma extensão homogênea, ele é implementado como o aspecto `Tracing.aj`, na linguagem AspectJ (Código 3.5) e atribuído ao diretório da *feature* `tracing`. A partir do momento em que um aspecto é atribuído a um módulo de *feature*, têm-se um módulo de característica aspectual.

Código 3.5 - Aspecto `Tracing.aj`

```
1 aspect Tracing{
2     after(): execution(public * Exemplo.*(..)) {
3         System.out.println("Imprime: " + variavel);
4     }
5 }
```

Esse aspecto é executado antes da chamada de qualquer método público da classe `Exemplo.jak`. Essa classe é implementada na linguagem Jakarta (Código 3.6) e atribuída ao módulo de *feature* denominado `Exemplo`. Essa classe possui o método `public void method()` (linhas 2 - 4), refinado pelos demais módulos de *features* da LPS. Dessa forma, como a classe `Exemplo` possui um método público refinado em todos os módulos de *features* da LPS (Código 3.7 - linhas 2 - 5), o aspecto `Tracing.aj` é executado depois de cada um dos refinamentos para mostrar a mensagem contendo informações a respeito do método. Além

disso, esse aspecto intercepta quaisquer outros métodos públicos implementados nos refinamentos da classe `Exemplo` (Código 3.7 - linha 7).

Código 3.6 - Classe `Exemplo.jak` no diretório `Exemplo`

```
1 public class Exemplo {
2     public void method(){
3         (..)
4     }
5 }
```

Código 3.7 - Refinamento da Classe `Exemplo.jak` nos demais Diretórios da LPS

```
1 public refines class Exemplo {
2     public void method(){
3         Super().method();
4         action();
5     }
6
7     public void action(){ (…) }
8 }
```

3.6.3 Comparação entre as Tecnologias para Extração de LPS

Nas tecnologias baseadas em anotação de código, são utilizadas diretivas para realizar a marcação do código a ser compilado. Na anotação textual, são adicionadas diretivas de pré-processamento no código para identificar os trechos de código pertencentes a cada *feature*. Contudo, as diretivas adicionadas ao código diminuem a sua legibilidade, o que pode tornar a manutenibilidade mais árdua. Para superar as limitações relacionadas à solução gerada e à legibilidade do código, as anotações visuais podem ser utilizadas, uma vez que garantem que as soluções geradas são compiláveis e fornecem um meio mais legível para realizar a anotação de código. No entanto, qualquer que seja a tecnologia baseada em anotação, a marcação de código não consiste em separar fisicamente os códigos das *features*, ou seja, os códigos marcados são combinados em tempo de compilação para gerar o novo produto de software.

Com as tecnologias baseadas em composição, é realizada a extração física de código. Com OA, pode-se modularizar os interesses transversais em aspectos, ou seja, as características espalhadas ou entrelaçadas pelo restante do sistema de software são separadas. Na OC, pode-se modularizar sistemas de software considerando as características como as principais abstrações. Por fim, foram apresentados MCA, nos quais há composição de aspectos e de características (*features*).

Quando comparadas, as tecnologias baseadas em anotação e em composição diferenciam-se quanto à implementação de *features* em granularidade fina ou grossa. As

extrações de *features* com tecnologias baseadas em composição é feita com granularidade grossa e com tecnologias baseadas em anotação com granularidade grossa ou fina. Além disso, as tecnologias baseadas em composição lidam de formas diferentes com interesses transversais, para os quais podem ser adicionadas extensões homogêneas ou heterogêneas. Na OA, há suporte para ambos os tipos de extensões, mas, para o tipo heterogêneo, o suporte é limitado. Na OC, há suporte apenas para extensões heterogêneas. Sendo MCA uma combinação de OA e de OC, há suporte para ambos os tipos de extensões. Em resumo, as diferenças entre as tecnologias baseadas em composição e em anotação são apresentadas na Tabela 3.1.

Tabela 3.1 - Comparação entre as Tecnologias de Composição e Anotação (Adaptado de Apel et al. (2008) e Liebig et al. (2010))

Critérios	Anotação		Composição		
	Anotação Textual	Anotação Visual	Orientação a Aspectos	Orientação a Características	Módulos de Características Aspectuais
Coloração de código	Não	Sim	Não	Não	Não
Criação de módulos	Não	Não	Sim	Sim	Sim
Diretivas de pré-processamento	Sim	Sim	Não	Não	Não
Granularidade grossa	Sim	Sim	Sim	Sim	Sim
Granularidade fina	Sim	Sim	Não	Não	Não
Extensão heterogênea	Sim	Sim	Parcial	Sim	Sim
Extensão homogênea	Não	Não	Sim	Não	Sim

3.7 Vantagens e Desvantagens

A abordagem LPS fornece benefícios classificados em (SILVA et al., 2011):

- **Benefícios Organizacionais**. Estão relacionados à capacidade que a empresa adquire para personalizar seus produtos em massa, manter a presença e continuar crescendo no mercado. De forma complementar, na pesquisa realizada pelo SEI (*Software Engineering Institute*) (NORTHROP; CLEMENTS, 2012), são citados os principais benefícios em nível organizacional, no qual há melhoria na produtividade em larga escala, na qualidade do produto, na agilidade do mercado, na satisfação do cliente e no uso eficiente dos recursos humanos. Além disso, há diminuição do tempo de mercado e do risco do produto;

- **Benefícios de Engenharia de Software**. Há melhor compreensão do domínio da aplicação, do controle de qualidade de software, do estabelecimento de padrões de programação e da reutilização de requisitos e seus componentes;

- **Benefícios de Negócio**. Há redução dos custos de manutenção e de teste. Além disso, fornece melhoria na eficiência nos processos e no planejamento do tempo, visto que tem maior controle dos componentes que fazem parte do produto final.

Na pesquisa do SEI (NORTHROP; CLEMENTS, 2012), é estabelecida uma relação de custo-benefício para os artefatos reutilizáveis da LPS. Para cada artefato, o custo de investimento, geralmente, é muito inferior ao valor do benefício. No entanto, é necessário ter gestão a longo prazo para os benefícios serem coletados, uma vez que são acumulados a cada nova versão do produto. Dessa forma, é necessário alto investimento inicial para o qual seria obtido um retorno a médio prazo. Além de necessitar de investimento inicial maior, LPS proporciona outras desvantagens. Para implantá-la em uma empresa, é necessário superar a resistência às mudanças (SILVA et al., 2011). Ainda, pode ser difícil estabelecer o escopo da LPS, visto que, se for muito amplo, torna complexa a reutilização dos artefatos, mas, se for muito pequena, não justifica o custo de desenvolvimento utilizando esta abordagem. Em resumo, as vantagens e as desvantagens na utilização de LPS são apresentadas na Tabela 3.2.

Tabela 3.2 - Vantagens e Desvantagens de LPS

Vantagens	Desvantagens
Maior personalização dos produtos Maior reutilização de artefatos Maior compreensão do domínio Maior lucratividade Maior qualidade do produto Maior presença de mercado Maior satisfação do cliente Menor volume de código Menor tempo de mercado Menor custo de manutenção e testes Menor risco do produto	Gestão a longo prazo Alto investimento inicial Resistência a mudanças na empresa Difícil estabelecer escopo da LPS

3.8 Considerações Finais

Neste capítulo, foi apresentada uma visão geral sobre LPS, a qual permite a customização em massa de produtos de software. Para o desenvolvimento de uma LPS são realizados os processos de Engenharia de Domínio e de Engenharia de Aplicação. Na Engenharia de Domínio são realizadas atividades para o desenvolvimento dos artefatos de software a serem reutilizados, ou seja, não há a derivação de um produto final. Diferentemente, na Engenharia de Aplicação, são realizadas atividades para reutilizar os artefatos de software de modo a derivar os produtos da LPS.

O principal artefato da LPS é o modelo de características, no qual as comunalidades e as variabilidades entre produtos da LPS são expressas por meio de *features* e seus relacionamentos. Esse modelo é utilizado como base para a infraestrutura destinada a

reutilização dos artefatos de software durante a derivação de produtos. Para o reúso efetivo dos artefatos de software é necessário que eles sejam modularizados para serem coerentes, bem definidos, independentes e combináveis. Nesse sentido, as tecnologias de gerenciamento de variabilidades podem ser utilizadas de modo a apoiar a modularização da LPS.

As tecnologias de gerenciamento de variabilidades são agrupadas em tecnologias baseadas em anotação e em composição. As tecnologias baseadas em anotação permitem a marcação do código das *features* por meio de anotações textuais ou visuais. Contudo, o código das *features* não são separados fisicamente. Por outro lado, as tecnologias baseadas em composição permitem a separação do código das *features* em módulos distintos. Dentre as diversas tecnologias baseadas em composição, são investigadas nesse trabalho: (i) Orientação a Aspectos, utiliza aspectos para modularizar os interesses transversais que afetam diferentes partes do sistema de software; (ii) Orientação a Características, realiza a modularização de sistemas de software considerando as características como principais abstrações; e (iii) Módulos de Características Aspectuais, realiza a combinação entre Orientação a Características e Orientação Aspectos.

4 MEDIDAS DE SOFTWARE EM LPS - ESTADO DA ARTE

4.1 Considerações Inicias

O conceito de qualidade de sistemas de software é subjetivo. A qualidade pode ser medida considerando a satisfação dos usuários, porém um sistema de software pode ser considerado de alta qualidade por alguns usuários e de baixa qualidade por outros. Desse modo, a qualidade pode ser considerada sob diferentes perspectivas (CÔTÉ et al., 2007). A qualidade sob a **perspectiva transcendental** é considerada como um ideal que pode nunca ser alcançado. Pela **perspectiva do usuário**, a qualidade é notada quanto à adequação do produto para o contexto de utilização. Pela **perspectiva do produto**, a qualidade considerada consiste nas características inerentes do produto. Na **perspectiva de fabricação**, a qualidade é medida em relação à conformidade dos requisitos. A **perspectiva final da qualidade** é baseada em valores.

Uma forma de aumentar as chances de produzir um sistema de software de qualidade é por meio da utilização de um processo de desenvolvimento de qualidade (CÔTÉ et al., 2007). A ideia é produzir, desde os estágios iniciais de desenvolvimento, artefatos de software de qualidade, o que pode resultar em um sistema de software final de qualidade. Para alcançar um processo de desenvolvimento de qualidade, podem ser utilizadas normas e modelos de qualidade, por exemplo, CMMI (*Capability Maturity Model Integration*) (TEAM, 2010), MPS.BR (Melhoria do Processo de Software Brasileiro) (SOFTEX, 2009) e ISO/IEC 33000:2015. Por outro lado, o produto resultante desse processo pode ser avaliado quanto à sua qualidade. Para isso, a norma ISO/IEC 25010:2011 pode ser utilizada para avaliar os atributos de qualidade de um produto/sistema de software.

Uma vez que diversos sistemas de software podem ser gerados a partir de uma LPS, esses sistemas podem ser impactados se seu *design* for de baixa qualidade (BAGHERI; GASEVIC, 2011). Desse modo, é necessário manter o controle da qualidade desde os estágios iniciais de desenvolvimento para evitar implicações de *design* de baixa qualidade em estágios mais avançados. Caso não haja essa preocupação inicial, pode resultar em desperdício de recursos e retrabalho para a construção de *design* de qualidade. Para aferir a qualidade, podem ser utilizadas medidas de software, pois fornecem indicação quantitativa de extensão, de quantidade, de dimensão, de capacidade ou de tamanho de algum atributo de um produto ou processo. Para tanto, é necessário que elas sejam facilmente entendidas e calculadas, de modo a fornecer melhoria no processo de desenvolvimento (SOMMERVILLE, 2016). Um detalhe importante é essas medidas serem propostas/adaptadas visando atender uma determinada

tecnologia. Por exemplo, as medidas utilizadas em sistemas de software orientados a objetos são diferentes das utilizadas em sistemas de software orientados a aspectos e em sistemas de software orientados a características.

Foi realizado um Mapeamento Sistemático da Literatura (MSL) que consistiu em buscar por medidas de software frequentemente utilizadas para mensurar as características de qualidade no contexto de LPS. Para isso, no restante desta seção, as questões de pesquisa, a definição e aplicação da *string* de busca nas bibliotecas digitais, a seleção e extração de dados dos artigos e os resultados obtidos para as questões de pesquisa são apresentados.

O restante deste capítulo está organizado da seguinte forma. Na Seção 4.2, são apresentadas as questões de pesquisa do MSL. Na Seção 4.3, é apresentado o processo de busca e seleção dos artigos. Na Seção 4.4, é apresentado o esquema de classificação e extração dos dados contidos nos artigos selecionados no MSL. Na Seção 4.5, os dados são sintetizados e apresentados de forma a responder as questões de pesquisa.

4.2 Questões de Pesquisa

Para o MSL, foram elaboradas quatro questões de pesquisa (QP), as quais são apresentadas juntamente com suas respectivas justificativas.

QP1: Quais atividades do ciclo de vida da LPS são frequentemente investigadas pela comunidade de pesquisa?

Justificativa. Identificar as atividades do ciclo de vida da LPS que são comumente investigadas pelos pesquisadores, o que pode proporcionar oportunidades de pesquisa nas atividades que possuem menos/mais investigação. Além disso, são identificadas as metodologias de pesquisa e as soluções frequentemente utilizadas para o estudo das atividades do ciclo de vida da LPS.

QP2: Quais características e subcaracterísticas de qualidade de produtos de software, definidas na ISO/IEC 25010:2011, são frequentemente investigadas no contexto de LPS?

Justificativa. Identificar as características e as subcaracterísticas de qualidade de produtos/sistemas de software, definidas na ISO/IEC 25010:2011, são comumente investigadas no contexto de LPS. Além disso, são identificados (i) o conjunto de propriedades de software utilizado para medir as subcaracterísticas, (ii) as medidas de software utilizadas para mensurar as propriedades de software e (iii) o paradigma de programação mais utilizado para desenvolver

uma LPS, fornecendo uma lista de medidas de software frequentemente utilizadas nos paradigmas de programação identificados.

QP3: Como o valor das medidas de software é extraído/obtido nos estudos?

Justificativa. Identificar o conjunto de dados utilizado pela comunidade acadêmica e as ferramentas e os sistemas de software mais utilizados para extrair/obter o valor das medidas de software.

4.3 Busca e Seleção de Artigos

Para responder as questões de pesquisa, foram selecionadas as bibliotecas digitais ACM[3], IEEE Xplore[4], Science Direct[5] e Scopus[6], pois atendem aos critérios de (i) pesquisa avançada utilizando palavras-chave, (ii) filtragem de resultados por ano de publicação e área de assunto, (iii) filtragem por tipo de publicação e (iv) exportação de resultados. Nessas bibliotecas, a busca abrangeu o período de 2000 a 2019. Cabe ressaltar que, inicialmente, a busca compreendeu o período de 2000 a 2018 e, posteriormente, a busca foi realizada no período de 2000 a junho de 2019 para a atualização dos dados obtidos. Com relação às opções de busca das bibliotecas digitais, foram incluídos `Journals`, `Magazines` e `Conference Proceedings`. Além disso, a busca foi limitada por área de assunto relacionada à `Computing` e os campos de busca para `Title`, `Abstract` e `Keywords`. Em seguida, a seguinte *string* de busca foi utilizada nas bibliotecas:

("product family" OR "family based" OR "product line")
AND
(metric OR measure)

Essa *string* de busca foi utilizada nessas bibliotecas, considerando a especificidade da máquina de busca (Tabela 4.1). Além disso, foram aplicados os critérios de inclusão e de exclusão para refinar os resultados da pesquisa apresentados na Tabela 4.2. A busca e a seleção dos artigos relevantes foram realizadas em cinco etapas (Tabela 4.3).

[3] Disponível em: https://dl.acm.org/dl.cfm
[4] Disponível em: http://ieeexplore.ieee.org/
[5] Disponível em: http://www.sciencedirect.com
[6] Disponível em: https://www.scopus.com/

Tabela 4.1 - *String* de Busca e Filtros Utilizados nas Bibliotecas

Bibliotecas	*String* de Busca
ACM Digital Library	recordAbstract:(("product family" OR "family based" OR "product line") AND (metric OR measure)) OR acmdlTitle:(("product family" OR "family based" OR "product line") AND (metric OR measure)) Filtros: "Published since: 2000" e "All Publications: Proceeding OR Periodical"
IEEE Xplore	(("product family" OR "family based" OR "product line")) AND (metric OR measure)) Filtros: "Content Types: Conferences, Journals and Magazines" e "Publication Year: from 2000 to present"
Science Direct	(("product family" OR "family based" OR "product line")) AND (metric OR measure)) Filtros: "Publication title: Journal of Systems and Software e Information and Software Technology", "Publication Year: from 2000 to 2019" e "Article type: Review articles e Research articles"
Scopus	(("product family" OR "family based" OR "product line")) AND (metric OR measure)) Filtros: "Document type: Article e Conference Paper", "Source type: Journal e Conference Proceedings", "Year: from 2000 to 2019", "Article type: Review articles e Research articles" e "Subject Area: Computer Science"

Tabela 4.2 - Critérios de Inclusão e de Exclusão

Sigla	Critério de Exclusão
E1	Artigos escritos em outro idioma, que não seja o inglês.
E2	Não-artigos (*e.g.*, normas e *table of contents*)
E3	A versão completa do artigo não está disponível *on-line*.
E4	Estudos Secundários

Sigla	Critério de Inclusão
I1	Artigos propondo/aplicando medidas de software para avaliar características de qualidade da LPS
I2	Artigos propondo/utilizando ferramentas para extração de medidas de software

Na primeira etapa (Tabela 4.3 - **Recuperados**), as bibliotecas digitais foram configuradas de acordo com o escopo da pesquisa em termos do período de tempo, da área de assunto, do tipo de documento e dos campos de busca. Como resultado, foram retornados 690 trabalhos, sendo 114 trabalhos da ACM (16,5%), 164 trabalhos da IEEE Xplore (23,8%), 20 trabalhos da Science Direct (2,9%) e 392 trabalhos da Scopus (56,8%); esses trabalhos foram armazenados em bases de dados distintas, ou seja, em arquivos diferentes. Na segunda etapa (Tabela 4.3 - **Filtro #1**), os resultados foram agrupados (armazenados em uma única base) e os trabalhos duplicados foram excluídos. Como resultado, 523 trabalhos permaneceram (75,8%), sendo 105 trabalhos da ACM (20,1%), 164 trabalhos da IEEE Xplore (31,4%), 20 trabalhos da Science Direct (3,8%) e 234 trabalhos da Scopus (44,7%). Na terceira etapa (Tabela 4.3 - **Filtro #2**), foram aplicados os critérios de exclusão. Como resultado, restaram 363 artigos[7] (69,4%), sendo 90 artigos da ACM (24,8%), 130 artigos da IEEE Xplore (35,8%), 19 artigos da Science

[7] Ressalta-se a alteração do termo "trabalho" para "artigo", pois "trabalho" = "artigo" + "não artigo", nesse contexto.

Direct (5,2%) e 124 artigos da Scopus (34,2%). Na quarta etapa (Tabela 4.3 - **Filtro #3**), foram aplicados os critérios de inclusão, em que apenas o título, o resumo e as palavras-chave dos artigos foram lidos. Como resultado, 99 artigos (27%) permaneceram, sendo 28 artigos da ACM (28,3%), 27 artigos da IEEE Xplore (27,3%), 9 artigos da Science Direct (9,1%) e 35 artigos de Scopus (35,3%). Na quinta etapa (Tabela 4.3 - **Selecionado**), foram aplicados os critérios de inclusão, em que o texto completo do trabalho foi lido. Como resultado, foi obtido o conjunto final com 60 artigos (60,6%), sendo 17 artigos da ACM (28,3%), 17 artigos da IEEE Xplore (28,3%), 7 artigos da Science Direct (11,7%) e 19 artigos da Scopus (31,7%).

Tabela 4.3 - Busca e Seleção de Artigos

Bibliotecas	Recuperados (Etapa 1)	Filtro #1 (Etapa 2)	Filtro #2 (Etapa 3)	Filtro #3 (Etapa 4)	Selecionados (Etapa 5)
ACM	114	105	90	28	17
IEEE Xplore	164	164	130	27	17
Science Direct	20	20	19	9	7
Scopus	392	234	124	35	19
Total	**690**	**523**	**363**	**99**	**60**

Posteriormente, foi definido um procedimento para a seleção dos estudos que envolveram o esforço de três pesquisadores (Pesquisador A, Pesquisador B e Pesquisador C). O Pesquisador A executou as atividades previstas para o MSL. O Pesquisador B e o Pesquisador C (experientes) auxiliaram o Pesquisador A. Esse procedimento consistiu em:

- O Pesquisador A aplicou a *string* de busca nas bibliotecas digitais e documentou o resultado da pesquisa[8];
- O Pesquisador A verificou o resultado da pesquisa, excluindo as duplicatas e os artigos que atendiam aos critérios de exclusão. Para artigos duplicados, foram mantidos os artigos da biblioteca digital com maior quantidade de palavras-chave indexadas[9];
- O Pesquisador A e o Pesquisador B verificaram os artigos, individualmente e separadamente, excluindo os artigos que não se enquadravam nos critérios de inclusão. A exclusão foi realizada após a leitura do título, resumo e palavras-chave. Os artigos foram excluídos apenas se fosse certo de que não se encaixavam nos critérios de inclusão;
- O Pesquisador A e o Pesquisador B realizaram a interseção dos artigos selecionados. No caso de discordância entre os pesquisadores, houve discussão sobre a inclusão do artigo. Se após a discussão os pesquisadores ainda não concordavam, então o artigo foi incluído;

[8] Documentação dos resultados disponível em: http://bit.ly/2YWyUh1
[9] Justificativa: Com mais palavras-chave indexadas, há melhor caracterização do artigo.

- O Pesquisador A e o Pesquisador C verificaram os artigos, individual e separadamente, excluindo os artigos que não se enquadravam nos critérios de inclusão. Eles excluíram os artigos após a leitura do texto completo;
- O Pesquisador A e o Pesquisador C realizaram a interseção dos artigos selecionados. No caso de discordância entre os pesquisadores, houve discussão sobre a inclusão do artigo. Como resultado, foi obtido o conjunto de artigos resultantes do mapeamento sistemático.

4.4 Esquema de Classificação e Extração dos Dados

Para coletar os dados dos artigos selecionados, na Tabela 4.4, é apresentada a relação entre os itens de dados e as questões de pesquisa. Cada item de dados é explicado a seguir:

- **Informações sobre a Publicação (D1)**. Foram coletados dados relacionados à publicação do artigo (título, autores, ano de publicação, local (periódico/evento) e editora). Esses dados foram usados para rastrear informações gerais sobre os artigos;
- **Atividade do Ciclo de Vida da LPS em que as Medidas são Utilizadas (D2)**. A qualidade pode ser mensurada em cada atividade do ciclo de vida da LPS. Desse modo, as atividades do ciclo de vida foram classificadas em (APEL et al., 2013): i) Engenharia de Domínio, composta pelas atividades Análise de Domínio, Arquitetura de Domínio, Implementação de Domínio e Teste de Domínio; e ii) Engenharia de Aplicação, composta pelas atividades Análise de Produto, Arquitetura de Produto, Implementação do Produto e Teste de Produto;
- **Proposta de Solução (D3)**. Os artigos foram classificados de acordo com a solução proposta para investigar o problema. Uma solução pode ser classificada como (CASTELLUCCIA; BOFFOLI, 2014): i) Visão geral; ii) Método; iii) Modelo; iv) Medida; e v) Ferramenta;
- **Metodologia de Pesquisa (D4)**. Os métodos de pesquisa dos artigos foram classificados quanto ao critério de avaliação (WIERINGA; ROLLAND, 2006): i) Pesquisa de Avaliação; ii) Proposta de Solução; iii) Pesquisa de Validação; iv) Trabalhos Filosóficos; v) Trabalhos de Opinião; e vi) Trabalhos de Experiência Pessoal;
- **Característica de Qualidade Avaliada (D5)**. O artigo foi classificado quanto à característica de qualidade avaliada. A classificação foi feita em termos da norma ISO/IEC 25010:2011, que possui as características de qualidade: i) Adequação Funcional;

ii) Eficiência de Desempenho; iii) Capacidade; iv) Usabilidade; v) Confiabilidade; vi) Segurança; vii) Manutenibilidade; e viii) Portabilidade;

- **Subcaracterística de Qualidade Avaliada (D6)**. Cada característica de qualidade da norma ISO/IEC 25010:2011 possui um conjunto de subcaracterísticas. Dessa forma, foram investigadas quais subcaracterísticas de qualidade foram avaliadas;
- **Propriedades de Software Mensuradas (D7)**. As medidas de software medem diversas propriedades dos artefatos de software. Desse modo, foram identificadas as propriedades de software mensuradas pelas medidas utilizadas nos artigos e sua relação com as (sub)características de qualidade;
- **Paradigmas de Programação Utilizados (D8)**. As medidas de software e as ferramentas utilizadas para a obtenção do seu valor estão ligadas ao paradigma de programação. Desse modo, foram identificados os paradigmas de programação utilizados nos artigos;
- **Nome dos Sistemas de Software Medidos (D9)**. Foi coletado o nome dos sistemas de software frequentemente utilizados para a avaliação da metodologia;
- **Nome das Ferramentas Utilizadas na Obtenção do Valor das Medidas (D10)**. Foi coletado o nome das ferramentas frequentemente utilizadas para obter o valor das medidas;
- ***Datasets* Utilizados para Obter os Valores das Medidas (D11)**. Foram coletados os *datasets* frequentemente utilizados para obter o valor das medidas;
- **Trabalhos Futuros (D12)**. A seção "Trabalhos Futuros" de cada artigo foi analisada para encontrar lacunas que os pesquisadores podem investigar.

Tabela 4.4 - Item de Dados

Item de Dados	Interpretação	QP
D1	Informação sobre a publicação	---
D2	Fase do ciclo de vida da LPS em que as medidas são usadas	QP1
D3	Proposta de solução	
D4	Metodologia de pesquisa	
D5	Característica de qualidade avaliada	QP2
D6	Subcaracterística de qualidade avaliada	
D7	Propriedades de software mensuradas	
D8	Paradigmas de programação utilizados	
D9	Nome dos sistemas de software mensurados	QP3
D10	Nome das ferramentas usadas na extração do valor das medidas	
D11	Datasets utilizados para obter o valor das medidas	
D12	Trabalhos futuros	---

4.5 Resultados

Ao todo, foram encontrados 20 artigos publicados em 10 periódicos diferentes (33,3%) e 40 artigos publicados em 25 eventos diferentes (66,7%). Na Tabela 4.5, são apresentados os periódicos com pelo menos dois trabalhos selecionados, sendo `Journal of Systems and Software` com maior quantidade de publicações (quantidade = 4 artigos). Na Tabela 4.6, são apresentados os eventos com, pelo menos, 2 artigos selecionados, sendo `International Conference on Software Product Lines` com maior quantidade de publicações (quantidade = 7 artigos). Mais informações sobre os artigos (título, autores, ano de publicação e biblioteca digital) podem ser encontradas no APÊNDICE A.

Tabela 4.5 - Periódicos com, pelo menos, 2 Artigos

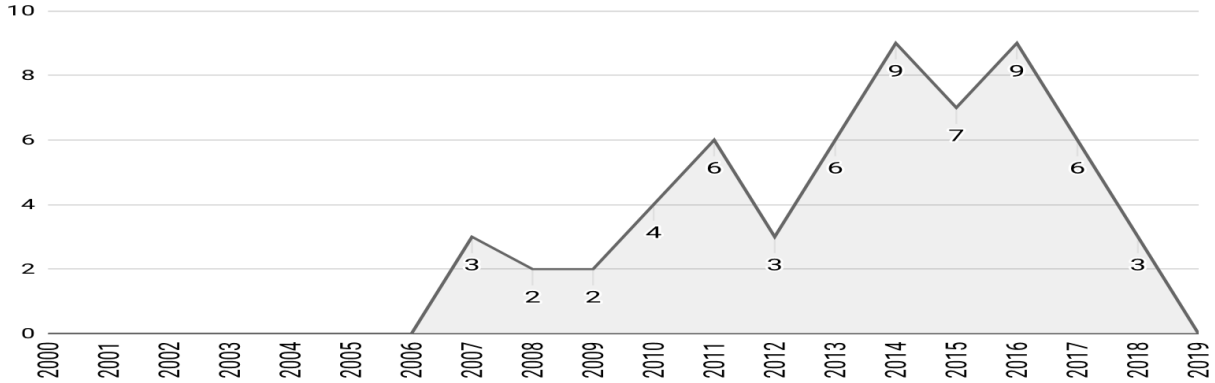
# Artigos	Periódico	Biblioteca
4	*Journal of Systems and Software*	Elsevier
3	*Information and Software Technology*	Elsevier
3	*Journal of Universal Computer Science*	IEEE
2	*Empirical Software Engineering*	Springer

Tabela 4.6 - Eventos com, pelo menos, 2 Artigos

# Artigos	Eventos	Biblioteca
7	*International Conference on Software Product Line*	ACM/IEEE
5	*Brazilian Symposium on Software Components, Architectures, and Reuse*	ACM/IEEE
3	*International Workshop on Variability Modelling of Software-Intensive Systems*	ACM
2	*Brazilian Symposium on Software Engineering*	IEEE
2	*European Conference on Software Maintenance and Reengineering*	IEEE
2	*International Conference on Software Engineering*	ACM
2	*International Conference on Software Testing, Verification and Validation*	IEEE
2	*International Workshop on Emerging Trends in Software Metrics*	ACM/IEEE
2	*International Workshop on Variability and Complexity in Software Design*	IEEE

Conforme os critérios de inclusão e exclusão, a busca por artigos compreendeu o intervalo de tempo entre 2000 e 2019 (inclusive). No entanto, no intervalo de tempo entre 2000 e 2006, os artigos retornados pelas bibliotecas digitais não foram considerados para o MSL, pois não atendiam os critérios de inclusão. Nos anos posteriores a esse período, houve ao menos uma publicação considerada relevante para o tema investigado (Figura 4.1). Pode-se perceber que o tema abordado no MSL teve quantidade crescente de publicações entre 2007 a 2017, mas a quantidade de publicações diminuiu em 2018 e, até o momento, não foram identificados artigos relevantes publicados em 2019.

Figura 4.1 - Distribuição Anual dos Artigos



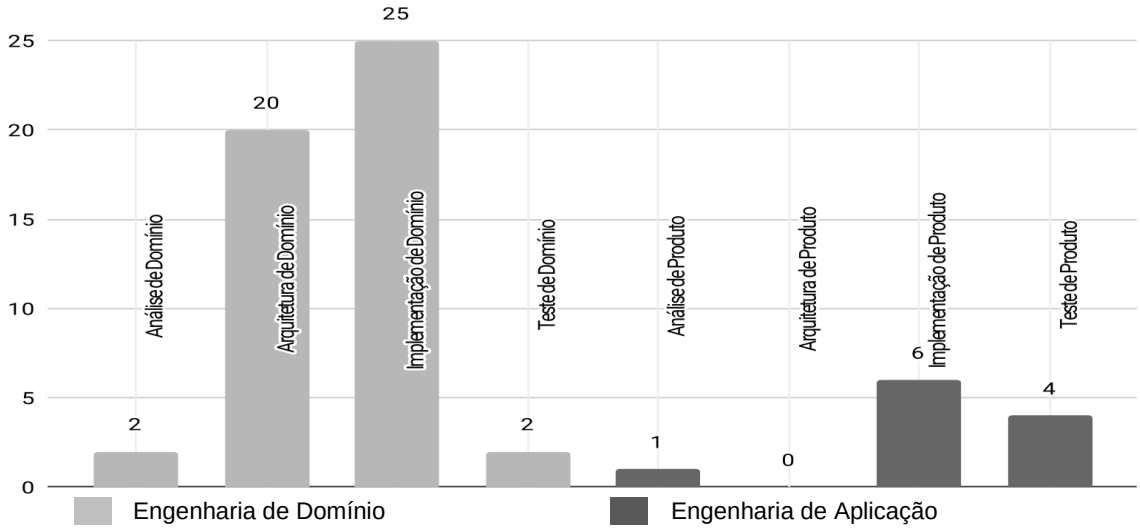
4.5.1 Investigação das Atividades do Ciclo de Vida da LPS

Em resposta a

QP1: Quais atividades do ciclo de vida da LPS são frequentemente investigadas pela comunidade de pesquisa?

dentre os artigos selecionados, o foco de 49 artigos (81,7%) é na investigação da Engenharia de Domínio e o foco de 11 artigos (18,3%) é na investigação da Engenharia de Aplicação (Figura 4.2). A obtenção de mais artigos na Engenharia de Domínio está alinhada à preocupação em desenvolver artefatos de software com qualidade desde estágios iniciais de desenvolvimento; caso contrário, a derivação de produtos da LPS pode ser impactada de forma negativa. A classificação dos artigos pode ser encontrada no APÊNDICE A.

Figura 4.2 - Quantidade de Artigos nas Atividades da Engenharia de Domínio e da Engenharia de Aplicação



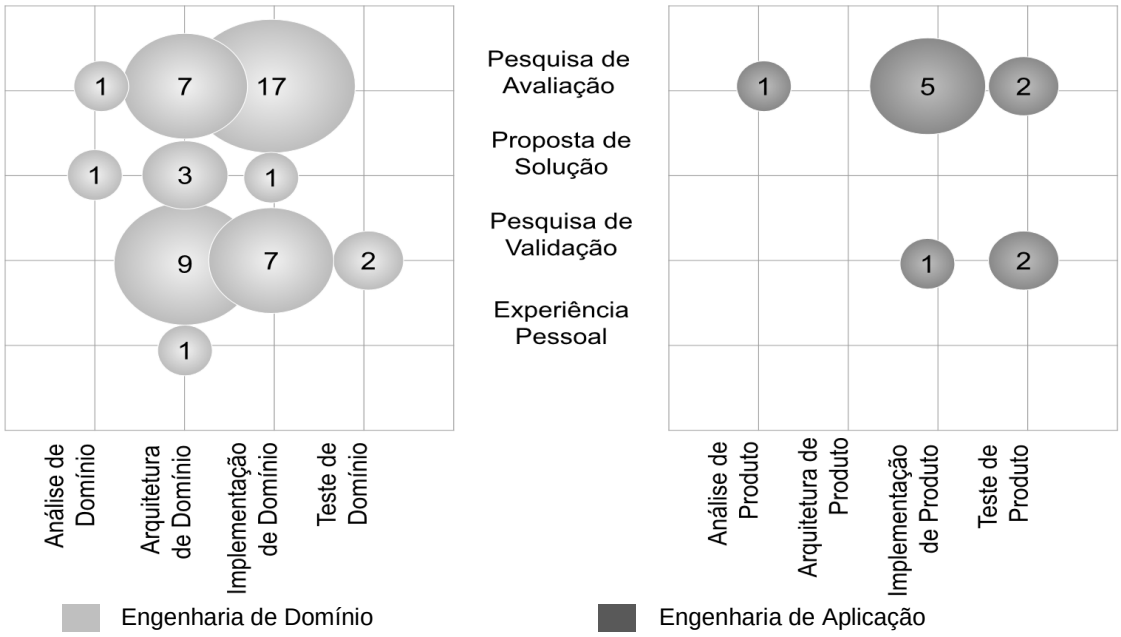
A Engenharia de Domínio é composta por 4 atividades. Quanto à atividade **Análise de Domínio**, que engloba atividades para elicitar, negociar, documentar, validar e gerenciar os

requisitos para o portfólio de produtos, foram encontrados 2 artigos (4,1%). Na atividade **Arquitetura de Domínio**, que engloba atividades para definir a arquitetura de referência da LPS, foram encontrados 20 artigos (40,8%). Na atividade **Implementação de Domínio**, que investiga o *design* detalhado e as técnicas de implementação dos artefatos de domínio, foram encontrados 25 artigos (51,0%). Na atividade **Teste de Domínio**, que investiga estratégias e técnicas de garantia de qualidade na presença de variabilidade, foram encontrados 2 artigos (4,1%). Portanto, as pesquisas focam na modelagem das variabilidades na arquitetura. Com esses resultados, pode-se perceber que o foco de vários trabalhos é na Implementação de Domínio, a fim de investigar propriedades relacionadas à arquitetura da LPS, como a sua estabilidade e sua evolução.

A Engenharia de Aplicação é composta por 4 atividades. Na atividade **Análise de Produto**, que investiga técnicas para definir requisitos para uma aplicação específica, foi encontrado 1 artigo (9,1%). Na atividade **Arquitetura de Produto**, que investiga maneiras de derivar uma arquitetura específica do aplicativo a partir da arquitetura de domínio, não foram encontrados artigos. Na atividade **Implementação de Produto**, que investiga a derivação de produtos, foram encontrados 6 artigos (54,5%). Na atividade **Teste de Produto**, que foca em testes dos produtos derivados, foram encontrados 4 artigos (36,4%). Portanto, o foco da pesquisa é a Implementação de Produtos, onde são investigadas quais técnicas permitem configurar, de forma consistente, os módulos do sistema.

Quanto à metodologia de pesquisa (Figura 4.3), foram classificados 33 artigos (55,0%) como **Pesquisa de Avaliação**, que investigam relações causais entre fenômenos, por meio de estudos de caso, estudos de campo, experimentos de campo e levantamentos. Como **Proposta de Solução**, que propõem uma técnica de solução e defende sua relevância com um pequeno exemplo, foram encontrados 5 artigos (8,3%). Como **Pesquisa de Validação**, que investigam as propriedades de uma proposta de solução por meio de experimentos e simulações, foram encontrados 21 artigos (35,0%). Apenas 1 artigo (1,7%) foi classificado como **Experiência Pessoal**, onde são relatadas as experiências provenientes de profissionais da indústria ou de pesquisadores que usaram suas ferramentas na prática. As metodologias de pesquisa mais utilizadas na Engenharia de Domínio foram a **Pesquisa de Avaliação**, cuja investigação é na Implementação do Domínio (17 artigos), e a **Pesquisa de Validação**, cuja investigação é na Arquitetura de Domínio (9 artigos) e no Teste Domínio (2 artigos). Além disso, a **Pesquisa de Avaliação** foi mais utilizada na Engenharia de Aplicação para investigar a Implementação do Produto (5 artigos). A classificação dos artigos quanto à metodologia de pesquisa utilizada pode ser encontrada no APÊNDICE A.

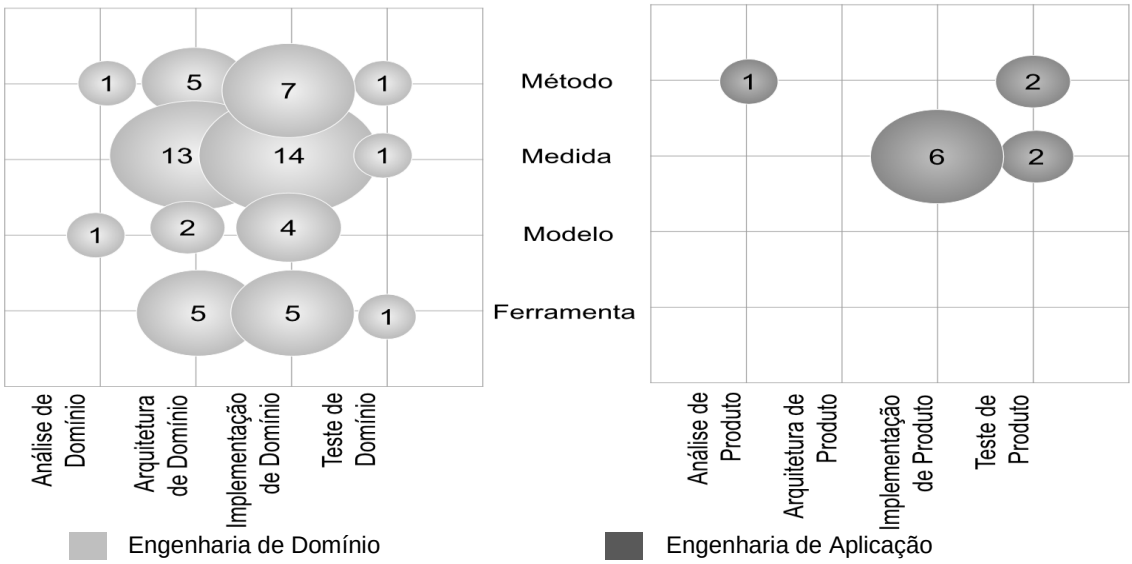
Figura 4.3 - Metodologia de Pesquisa dos Artigos



Quanto à solução proposta (Figura 4.4), foram encontrados 17 artigos (23,9%) propondo **Método** como solução, que consiste no fluxo de trabalho de atividades, nas regras ou nos procedimentos sobre como as coisas devem ser feitas. Outros 36 artigos (50,7%) propuseram **Medida** como solução, que descreve novas medidas, um plano contendo um conjunto de medidas para mensurar e avaliar uma LPS. A solução **Modelo** foi proposta em 7 artigos (9,9%), que descreve uma representação conceitual, com abstração formal de detalhes e de notações. A solução **Ferramenta** foi proposta em 11 artigos (15,5%), sendo utilizada para apoiar as outras categorias de solução, especificamente, em 6 artigos para apoiar a solução Medida (54,5%), em 3 artigos para apoiar a solução Método (27,3%) e em 2 artigos para apoiar a solução Modelo (18,2%). A solução Medida foi a mais proposta para a Arquitetura de Domínio (13 artigos), Implementação de Domínio (14 artigos) e Implementação de Produtos (6 artigos). Mais informações sobre a solução proposta nos artigos podem ser encontradas no APÊNDICE A.

Cabe ressaltar que, conforme a classificação de metodologia e solução (WIERINGA; ROLLAND, 2006; CASTELLUCCIA; BOFFOLI, 2014), não foram encontrados nos artigos a metodologia Pesquisa Filosófica e a solução *Overview*. Elas são típicas de estudos secundários (mapeamento sistemático e revisão sistemática). Dessa forma, os trabalhos que se encaixavam nessas categorias foram removidos durante a aplicação dos critérios de exclusão (Tabela 4.3 - **Filtro #2**), visto que, em MSL, são considerados apenas trabalhos primários.

Figura 4.4 - Soluções Propostas nos Artigos



4.5.2 Mensuração da Qualidade de Produtos de Software com a ISO/IEC 25010

Em resposta a questão

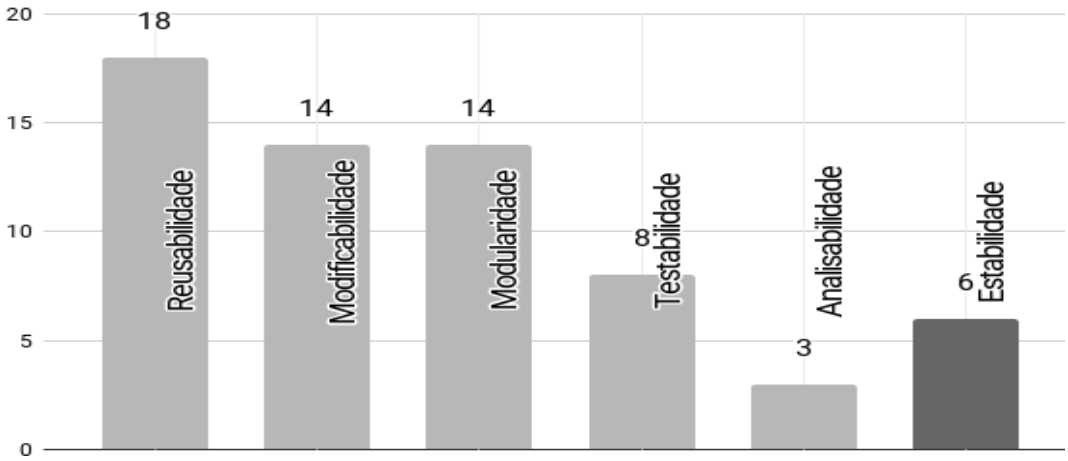
QP2: Quais características e subcaracterísticas de qualidade de produtos de software, definidas na ISO/IEC 25010:2011, são frequentemente investigadas no contexto de LPS?

foram identificadas três características de qualidade relevantes para a comunidade de pesquisa: i) Manutenibilidade; ii) Confiabilidade; e iii) Eficiência do Desempenho. A Manutenibilidade foi investigada por 51 artigos (85,0%); sendo assim, essa característica pode ser considerada com mais relevância no contexto de LPS. Na sequência, foram encontrados 5 artigos (8,3%) avaliando a característica Confiabilidade e 4 artigos (6,7%) avaliando a característica Eficiência de Desempenho. Quanto aos artigos que avaliaram a característica Confiabilidade, apenas a subcaracterística Maturidade foi identificada; com relação aos artigos que avaliaram a característica Eficiência do Desempenho, apenas a subcaracterística Comportamento Temporal foi identificada. Para a característica Manutenibilidade, uma ou mais de suas subcaracterísticas foram avaliadas em um artigo (Figura 4.5).

Quanto à avaliação da característica Manutenibilidade (Figura 4.5), foram avaliadas as subcaracterísticas: i) Modificabilidade (14 artigos - 22,2%); ii) Testabilidade (8 artigos - 12,7%); iii) Analisabilidade (3 artigos - 4,8%); iv) Reusabilidade (18 artigos - 28,6%); e v) Modularidade (14 artigos - 22,2%). Além disso, como a norma ISO/IEC 25010:2011 surgiu em 2011 e a busca por artigos compreendeu o período de 2000 a 2019, foram encontradas referências à norma ISO/IEC 9126 que apresenta a subcaracterística Estabilidade (capacidade

do sistema de software evitar efeitos colaterais por causa de modificações realizadas). Na norma ISO/IEC 25010:2011, a subcaracterística Estabilidade evoluiu para subcaracterística Modificabilidade, representando a combinação entre essas subcaracterísticas. Dessa forma, nesse MSL, a subcaracterística Estabilidade foi considerada uma subcaracterística de Manutenibilidade à parte, sendo avaliada em 6 artigos (9,5%). De modo geral, a subcaracterística Reusabilidade foi a mais avaliada nos artigos, seguida pelas subcaracterísticas Modularidade e Modificabilidade.

Figura 4.5 - Ocorrências das Subcaracterísticas de Manutenibilidade



Além disso, uma subcaracterística de qualidade pode ser mensurada quanto a uma ou mais propriedades de software. Na Tabela 4.7, é resumida a relação entre 15 propriedades do software encontradas e as subcaracterísticas de qualidade de Manutenibilidade. Por exemplo, para a subcaracterística Modularidade, são analisadas as propriedades de software acoplamento, coesão, complexidade, herança, tamanho, entrelaçamento e espalhamento. Isso significa que, diferentes trabalhos analisaram essas propriedades de modo a aferir o quanto elas impactam (positiva ou negativamente) a Modularidade. Uma LPS que possui baixo acoplamento e alta coesão entre seus módulos (*features*) apresenta boa modularidade. De modo contrário, o alto nível de herança, de entrelaçamento e de espalhamento pode indicar alto acoplamento e a alta complexidade e tamanho pode indicar a baixa coesão nos módulos (*features*) da LPS.

Para mensurar essas propriedades, podem ser utilizadas medidas de software. Ao todo, foram identificadas 341 medidas de software, sendo 13 medidas relacionadas às características Eficiência de Desempenho e Confiabilidade (3,8%) e 328 medidas relacionadas à característica Manutenibilidade (96,2%). Como as medidas de Eficiência de Desempenho e de Confiabilidade são destinadas ao comportamento do produto, elas não foram relacionadas às propriedades de software. Dessa forma, na **Figura 4.6**, é apresentada a distribuição da quantidade de medidas

relacionadas às propriedades de software identificadas para mensurar as subcaracterísticas de Manutenibilidade. Uma medida pode mensurar mais de uma propriedade de software, de modo que, ao contar de forma única uma medida para uma propriedade, foram obtidas 381 medidas. Além disso, na Tabela 4.8, são listadas as medidas de software referenciadas por, ao menos, 2 artigos, para mensurar as 5 propriedades de software mais investigadas (Tamanho, Acoplamento, Complexidade, Coesão e Comunalidade). Mais informações sobre as medidas podem ser encontradas no APÊNDICE B .

Tabela 4.7 - Relação entre as Subcaracterísticas de Qualidade de Manutenibilidade e as Propriedades de Software

Características / Propriedades	ISO/IEC 25010					ISO/IEC 9126
	Analisabilidade	**Modificabilidade**	**Modularidade**	**Reusabilidade**	**Testabilidade**	**Estabilidade**
Abstração	X	X			X	
Acoplamento	X	X	X	X		X
Clone		X		X		
Code Churn		X		X		
Code Smell		X				
Coesão	X	X	X	X		X
Complexidade	X	X	X	X	X	X
Comunalidade				X		X
Entrelaçamento		X	X	X		X
Espalhamento		X	X	X		X
Herança	X		X	X	X	X
Mensagem				X		
Polimorfismo				X	X	
Similaridade				X		X
Tamanho	X	X	X	X	X	X

Figura 4.6 - Quantidade de Medidas Relacionadas às Propriedades de Software

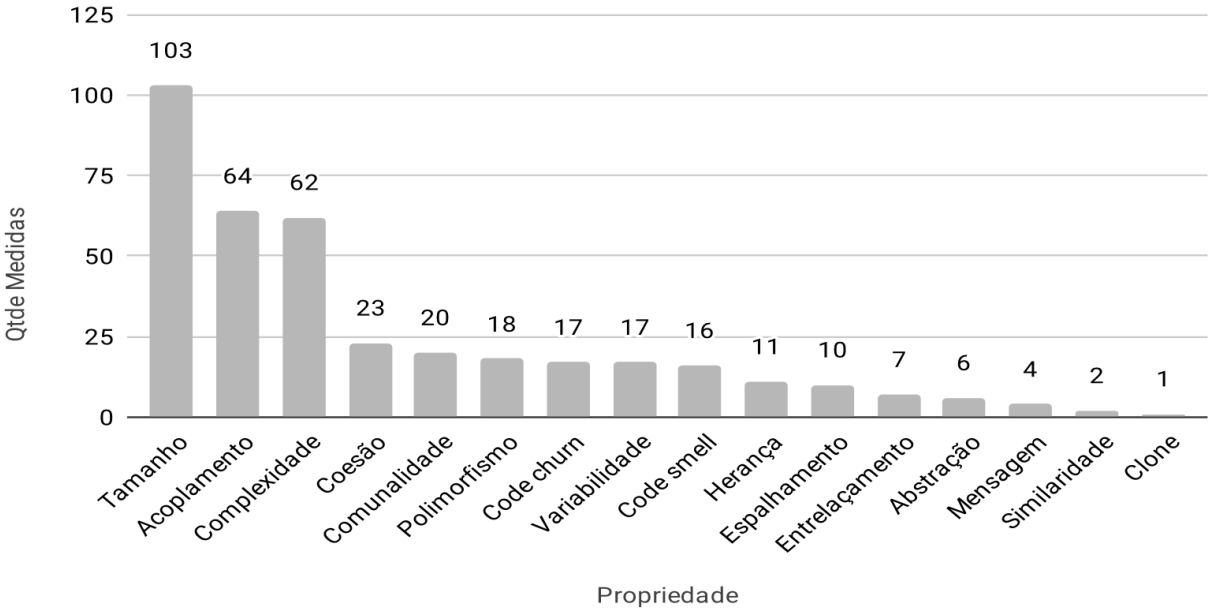


Tabela 4.8 - Medidas Utilizadas nas 5 Propriedades de Software mais Investigadas

Propriedades	Medidas	# Artigos
Tamanho	*Lines of Code* (LOC)	16
	Number of Features (NF)	8
	Number of Leaf Features (NLeaf)	6
	Number of Constant Refinements (NCR) *Number of Top Features* (NTop) *Source Lines of Code* (SLOC)	5
	Lines of Feature Code (LOF) *Number of Components* (NOC) *Vocabulary Size* (VS)	4
	Number of Classes (NCLASS) *Number of Mandatory Features* (NM)	3
	Cross-Tree-Constraints Ratio (CTCR) *Lines of Concern Code* (LOCC) *Lines of Feature Code* (LOF) *Number of Features Constants* (NOFC) *Number of Methods* (NOM) *Number of Operations* (NOO) *Number of OR relationship* (Ror) *Number of Products* (NP) *Number OR Groups* (NGOr) *Number XOR Groups* (NGXor) *XOr Feature Ratio* (RXor)	2
Acoplamento	*Coupling Between Objects* (CBO)	5
	Afferent Coupling (Ca) *Efferent Coupling* (Ce) *Coupling Between Components* (CBC) *Structural Feature Coupling* (SFC)	4
	Component-level Interlacing Between Features (CIBC) *Feature EXtendibility* (FEX) *Multiple Cyclic Dependent Features* (MCDF) *Multiple Hotspot Features* (MHoF) *Single Cyclic Dependent Features* (SCDF)	3
	Class fragmentation (ClassFrag) *Feature-reference graph degree* (Ref) *Instability* (I) *Operation-level Overlapping Between Features* (OOBC)	2
Complexidade	*Number of Features* (NF)	8
	Cyclomatic Complexity (CyC)	7
	Number of Methods Annotated (NFM) *Number of Leaf Features* (NLeaf)	6
	Density of the graph (Rden) *Flexibility of Configuration* (FoC) *Number of Top Features* (NTop) *Number of Valid Configurations* (NVC) *Ratio of Variability* (RoV)	5
	Cognitive Complexity (CogC) *Wheighted Methods per Class* (WMC)	4
	Coefficient of Connectivity-Density (CoC) *Compound Complexity* (ComC) *Cross-Tree Constraints* (CTC) *Feature EXtendibility* (FEX) *Maximum Depth of Tree* (DTMax) *Response for a Class* (RFC)	3
	Average Block Depth *Average Complexity* *Average Nesting Depth* CompClass CompPLA *Maximum Complexity* *Mean Depth of Tree* (DTMean) *Median Depth of Tree* (DTMedian) *Number of grouping features* (NGF) *Ratio of connectivity of this graph* (RCon)	2

Tabela 4.8 - Medidas Utilizadas nas 5 Propriedades de Software mais Investigadas (cont.)

Propriedades	Medidas	# Artigos
Coesão	*Internal-ratio Feature Dependency (IFD)* *External-ratio Feature Dependency (EFD)* *Lack of Cohesion in Operations (LCO)* *Lack of Feature-based Cohesion (LCC)* *Lack of Cohesion in Methods (LCOM)*	2
Comunalidade	*Generational Variety Index (GVI)* *Product line commonality index (PCI)*	2

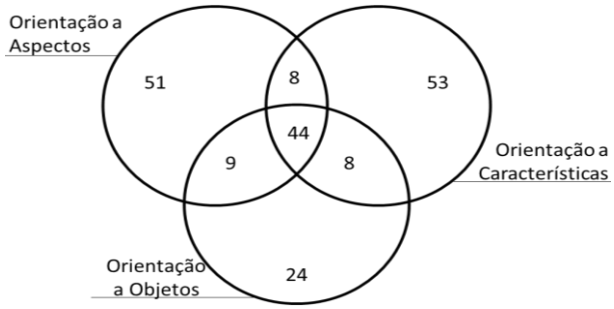
Em relação ao objeto de mensuração, foi identificado que, das medidas utilizadas para avaliar a Manutenibilidade (328 medidas), 131 medidas são destinadas à análise de sua arquitetura (39,9%) e 197 medidas são destinadas à análise de seu código fonte (60,1%). Nesse sentido, foram identificados 2 mecanismos relacionados à arquitetura, sendo 2 medidas utilizadas em ambos mecanismos:

- **Modelo de características**. Foram especificadas medidas a serem utilizadas no modelo de características FODA. Foram encontradas 69 medidas (67 medidas + 2 medidas comuns ao meta-modelo). A medida *Number of Leaf Features* (NLeaf) foi a mais referenciada (6 artigos);
- **Meta-modelo**. Foram especificadas medidas a serem utilizadas considerando a arquitetura do sistema, utilizando diagramas de classes/pacotes. Ao todo, foram encontradas 64 medidas (62 medidas + 2 medidas comuns ao modelo de características). A medida *Number of Classes* (NCLASS) foi a mais referenciada (3 artigos).

Quanto aos paradigmas de programação (Figura 4.7), foram identificas 197 medidas utilizadas na análise de código fonte em:

- **Orientação a Aspectos**. Ao todo, foram coletadas 51 medidas (25,8%) utilizadas exclusivamente em OA. As medidas *Degree of Scattering accross Components* (DOSC), *Degree of Tangling within Components* (DOTC), *Pairs of Cloned Code* (PCC) e *Vocabulary Size* (VS) foram as mais referenciadas (4 artigos cada);
- **Orientação a Características**. Ao todo, foram encontradas 53 medidas (26,9%) utilizadas exclusivamente em OC. A medida *Number of Constant Refinements* (NCR) foi a mais referenciada (5 artigos);
- **Orientação a Objetos**. Ao todo, foram encontradas 24 medidas (12,2%) utilizadas em OO. A medida *Cyclomatic Complexity* (CyC) foi a mais referenciada (7 artigos).

Figura 4.7 - Distribuição das Medidas nos Paradigma de Programação



Além disso, ocorre intersecção entre os paradigmas identificados. Geralmente, as medidas OO foram adaptadas para o contexto de LPS, uma vez que elas são amplamente difundidas para a mensuração de código fonte (*e.g.*, medidas de CK e medidas de QMOOD). Na Tabela 4.9, são apresentadas as medidas referenciadas em, pelo menos, dois artigos, conforme os paradigmas em que ocorreram intersecção. Cabe destacar que as medidas utilizadas em OA e OC podem ser aplicadas em MCA, visto que utilizam os mesmos princípios.

Tabela 4.9 - Medidas para Diversos Paradigmas de Implementação da LPS

Intersecção	Ocorrências	Medidas
OO e CO	2	*Feature Diffusion over Architectural Components* (CDAC) *Feature Diffusion over Architectural Interfaces* (CDAI) *Feature Diffusion over Architectural Operations* (CDAO) *Interface-level Interlacing Between Features* (IIBC)
OO e OA	4	*Coupling Between Components* (CBC)
	2	*Instability Metric* (MI)
OA e OC	7	*Scattering Degree* (SD) *Tangling Degree* (TD)
OA, OC e OO	16	*Lines of Code* (LOC)
	7	*Cyclomatic Complexity* (CC / CyC)
	2	*Lack of Feature-based Cohesion* (LCC) *Internal-ratio Feature Dependency* (IFD) *External-ratio Feature Dependency* (EFD) *Lack of Cohesion in Operations* (LCO)

4.5.3 Obtenção de Valores para as Medidas de Software

Em resposta a questão

QP3: Como o valor das medidas de software é extraído/obtido nos estudos?

foram identificados 3 conjuntos de dados: i) **AFFOgaTO** (*dAtaset For the Feature mOdel evoluTiOn*); ii) **ESPREssO** (*mEasures dataSet for dsPl featuRE mOdel*); e iii) **MAcchiATO** (*MeAsures dATaset for feaTure mOdel*). Cada conjunto de dados foi construído com ferramenta `Dymmer`, a partir da qual foram extraídas as medidas estruturais dos modelos de características do repositório SPLOT (*Software Product Lines Online Tools*).

Além disso, as medidas foram coletadas de 141 sistemas de software utilizados como *benchmarks*, sendo 90 sistemas de software acadêmicos (63,8%), 44 sistemas de software de código aberto (31,2%) e 7 sistemas de software de código proprietário (5,0%). Os sistemas do tipo Acadêmico são encontrados em repositórios (por exemplo, SPLOT[10], SPL2go[11] e FeatureIDE[12]) ou foram desenvolvidos para apoiar pesquisas. Os sistemas das categorias acadêmica e código aberto são preferidos pela comunidade de pesquisa para a realização de *benchmarks*. O sistema de software `BerkeleyDB` é o mais utilizados como *benchmark* acadêmico (11 artigos) e o sistema de software `Freemind` (4 artigos). Outros sistemas de software utilizados podem ser encontrados no APÊNDICE B.

Além disso, foram encontradas 12 ferramentas usadas para obter o valor das medidas dos sistemas de software. Na Tabela 4.10, é apresentada a quantidade de ocorrências das ferramentas. A maioria dos pesquisadores propôs uma ferramenta específica para coletar dados da LPS. No entanto, como essas ferramentas não foram a principal contribuição do artigo nem foram disponibilizadas em domínio público, elas não foram contabilizadas.

Tabela 4.10 - Ferramentas para Obtenção do Valor das Medidas em LPS

Ocorrências	Ferramentas
4	Dymmer
3	CodePro Analitix Metrics SourceMonitor
2	AOP-Metrics CCFinder CKJM JHawk OPLA-Tool
1	FLOrIDA MT TDTool

4.6 Discussão

Em relação as lacunas e as tendências de pesquisa, pode-se perceber que a característica de qualidade Manutenibilidade é a mais explorada no contexto de LPS; especialmente, em relação à subcaracterística Reusabilidade. Isso é justificável pelo fato da abordagem LPS ser baseada no reúso sistemático de artefatos de software. Nesse sentido, no artigo P3, é descrito um *framework* adaptado a partir da norma ISO/IEC 9126 para avaliar os artefatos de software da LPS quanto à Reusabilidade (HER et al., 2007). Os autores justificaram que a ISO/IEC 9126 avalia um único produto, sendo insuficiente para avaliar de forma precisa a reutilização dos

[10] Disponível em: <http://www.splot-research.org/>
[11] Disponível em: <http://spl2go.cs.ovgu.de/>
[12] Disponível em: <https://featureide.github.io/>

artefatos de software da LPS. Similarmente, a norma ISO/IEC 25010:2011 poderia ser estudada para a derivação de um *framework* geral destinado à avaliação da qualidade em LPS.

Além disso, a interpretação do valor de medidas de software não é trivial. Dessa forma, é importante que os *thresholds* sejam fornecidos para auxiliar o processo de interpretação das medidas. No artigo P24, é fornecido um método para derivar os *thresholds* das medidas no contexto de LPS. Assim, os estudos que propõem novas medidas de software no contexto de LPS poderiam usar esse método para derivar os *thresholds* e avaliar a aplicabilidade das medidas de software.

Adicionalmente, a repetibilidade das medidas entre os artigos é baixa, o que mostra a baixa disseminação de um conjunto de medidas de software pela comunidade de pesquisa. Dessa forma, um conjunto de medidas e seus valores limites poderiam ser propostos e avaliados para medir diferentes propriedades de software (*e.g.*, medidas QMOOD). No artigo P30, é apresentado um conjunto de medidas e seus valores limites para Arquitetura de Domínio, utilizando o modelo de características. Esse estudo poderia ser estendido para as outras atividades do ciclo de vida da engenharia de LPS.

4.7 Ameaças a Validade

As ameaças à validade são organizadas em quatro tipos (TRAVASSOS *et al.,* 2002): (i) Validade Interna; (ii) Validade Externa; (iii) Validade de Constructo; e (iv) Validade de Conclusão. A seguir são detalhadas as ameaças a validade do MSL.

- **Validade Interna.** Essa ameaça relaciona-se com os fatores que afetam o processo de experimentação, sendo o resultado casual (TRAVASSOS et al., 2002). A principal limitação está relacionada ao processo de busca, em que novas palavras-chave poderiam ter sido adicionadas a *string* de busca após a maturação do tema. Além disso, a busca foi realizada apenas de forma automática, não sendo buscados artigos relevantes de forma manual.

- **Validade Externa.** Essa ameaça relaciona-se com a capacidade de generalização da descoberta (TRAVASSOS et al., 2002). Essa ameaça está relacionada a forma de análise das medidas, visto que em diversos trabalhos não foram fornecidas definições matemáticas.

- **Validade de Construto.** Essa ameaça refere-se aos fatores que afetam o processo de experimentação, em que o resultado reflete o efeito (TRAVASSOS et al., 2002). O MSL

foi realizado de modo a responder quatro questões de pesquisa, no entanto podem haver outras questões de pesquisa relevantes a serem investigadas.

- **Conclusão Validade.** Essa ameaça refere-se à capacidade de obter a uma conclusão correta em relação aos fatores que afetam o processo de experimentação e o resultado do experimento (TRAVASSOS et al., 2002). A partir da realização do MSL foram obtidos 60 artigos, dos quais foram coletadas 341 medidas. Como muitos trabalhos não descrevem matematicamente como calcular as medidas de software utilizadas, algumas delas podem ser idênticas e diferir apenas pela abreviação. Dessa forma, no MSL todas as medidas que não puderam ser diferenciadas, foram relatadas como uma nova medida.

4.8 Considerações Finais

Nesse capítulo, foi descrito o MSL realizado para identificar as medidas de software frequentemente utilizadas para mensurar as características de qualidade em LPS. A partir dos dados identificados no MSL, foi observado que a característica de qualidade mais investigada no contexto de LPS é a Manutenibilidade e, especialmente, quanto às suas subcaracterísticas Reusabilidade, Modificabilidade e Modularidade.

Dado que este trabalho propõe uma investigação quanto à modularidade de uma LPS desenvolvida com três tecnologias de gerenciamento de variabilidades, foram observadas as propriedades de software mais investigadas no contexto de LPS. Como resultado, foi identificado que, para mensurar a modularidade em LPS, são utilizadas as propriedades Tamanho, Complexidade, Entrelaçamento, Espalhamento, Acoplamento e Coesão. As propriedades Tamanho e Complexidade estão relacionadas à coesão dos módulos, pois são referentes as “responsabilidades” atribuídas a eles. As propriedades Entrelaçamento e Espalhamento estão relacionados ao acoplamento entre os módulos, pois indicam relação de dependência entre eles. Dessa forma, a modularidade da LPS pode ser mensurada em termos das propriedades de acoplamento e de coesão.

5 SPLiME - UM APOIO COMPUTACIONAL PARA EXTRAÇÃO DE MEDIDAS EM LINHAS DE PRODUTOS DE SOFTWARE

5.1 Considerações Inicias

Na medição de código, um dos objetivos é avaliar a qualidade interna de um sistema de software. Nesse trabalho, foi desenvolvido um apoio computacional (*plug-in* para o Eclipse IDE) para coletar medidas de uma LPS desenvolvida com três diferentes tecnologias: i) Orientação a Características, na linguagem Jakarta; ii) Orientação a Aspectos, na linguagem AspectJ; e iii) Módulos de Características Aspectuais, com as linguagens Jakarta e AspectJ.

O desenvolvimento do *plug-in* `SPLiME` (*Software Product Line Maintainability Evaluation*) é justificado pela dificuldade em encontrar ferramentas automatizadas para mensurar sistemas desenvolvidos em AspectJ e/ou Jakarta (REIS et al., 2014). Diversas ferramentas, por exemplo Eclipse Metrics[13] e Analizo[14], calculam medidas de software na linguagem Java; em especial, medidas voltadas para a tecnologia Orientação a Objetos. No entanto, para as linguagens Jakarta e AspectJ, extensões da linguagem Java, essas ferramentas não oferecem suporte adequado para a análise do código da LPS e as medidas não abordam propriedades no contexto de LPS, tais como, entrelaçamento e espalhamento de interesses pelos módulos do sistema. Outras ferramentas como AOPMetrics[15] e MT[16], calculam medidas de software na linguagem AspectJ e Jakarta, respectivamente. No entanto, essas ferramentas calculam diferentes conjuntos de medidas e, além disso, não há garantia de uma mesma medida implementada em ambas as ferramentas seja calculada considerando as mesmas informações. Dessa forma, foi necessário o desenvolvimento de um apoio computacional de modo a calcular as medidas relacionadas à modularidade da LPS de forma similar, utilizando essas linguagens.

O restante deste capítulo está organizado da seguinte forma. Na Seção 5.2, é apresentado o conjunto de medidas de software selecionadas para avaliar a modularidade da LPS, quanto ao acoplamento e à coesão de *features*. Na Seção 5.3, é apresentado o processamento do `SPLiME` em alto nível e sua interface de interação com o usuário. Na Seção 5.4, é apresentada a organização do `SPLiME` e detalhes de sua implementação.

[13] Disponível em: <https://sourceforge.net/projects/metrics/>
[14] Disponível em: <http://www.analizo.org/>
[15] Disponível em: < https://github.com/ozlerhakan/aop-metrics>
[16] Disponível em: < http://labsoft.dcc.ufmg.br/doku.php?id=about:feature-oriented_measures_to_assess_software_product_lines>

5.2 Medidas Selecionadas

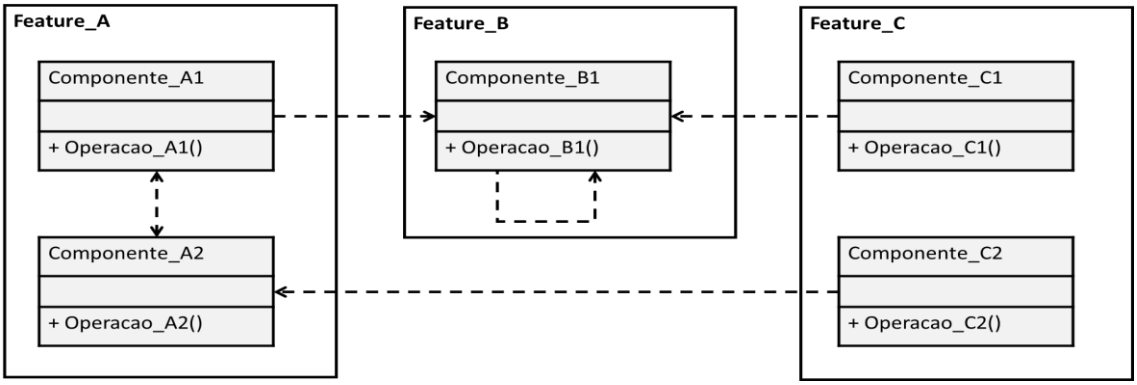
A seleção das medidas de acoplamento e de coesão foi realizada com base nos dados apresentados no Capítulo 4. Em relação a propriedade de coesão, foram identificadas as medidas *External-ratio Feature Dependency* (EFD), *Internal-ratio Feature Dependency* (IFD), *Lack of Concern-based Cohesion* (LCC), *Lack of Cohesion in Methods* (LCOM) e *Lack of Cohesion in Operations* (LCO). Em relação a propriedade de acoplamento, foram identificadas as medidas *Afferent Coupling* (Ca), *Efferent Coupling* (Ce), *Coupling Between Components* (CBC), *Coupling Between Objects* (CBO) e *Structural Feature Coupling* (SFC).

Inicialmente, a seleção das medidas consistiu em verificar se poderiam ser utilizadas em OA e em OC. Nesse momento, foram buscadas medidas alternativas, talvez não muito citadas, para as medidas desenvolvidas em OO. Para as medidas CBO e LCOM, pertencentes ao conjunto de medidas de CK (CHIDAMBER; KEMERER, 1994), foram encontradas alternativas dentre as medidas mais citadas, CBC e LCOO, respectivamente. Para as medidas Ca e Ce, pertencentes ao conjunto de medidas de Martin (MARTIN, 1994), não foram encontradas alternativas dentre as medidas mais citadas. No entanto, foram encontradas as medidas *Dependency In* (DepIn) e *Dependency Out* (DepOut), respectivamente. Como resultado, restaram as medidas de coesão EFD, IFD, LCC e LCO e as medidas de acoplamento DepIn, DepOut, CBC, SFC.

Posteriormente, foi definido que as propriedades de acoplamento e de coesão seriam calculadas em nível de *feature*. Com isso, as medidas LCC, LCO e CBC foram analisadas a fim de estabelecer possíveis adaptações para serem aplicadas. As medidas LCC e CBC são calculadas em nível de componente e a medida LCO em nível de operações. A medida CBC foi excluída, pois ela é dada pela soma das dependências de entrada e de saída de um componente. Desse modo, ao considerar o nível de *feature*, a medida seria dada pela soma das medidas DepIn e DepOut. A medida LCC foi mantida, pois ela calcula a coesão de um componente com base na quantidade de *features* que ele implementa. Então, em nível de *feature*, bastaria verificar se os componentes de uma *feature* implementam outras *features*. A medida LCO foi excluída, pois as mudanças seriam mais custosas, visto que seria preciso modificar a medida de nível de operações para nível de componentes e, posteriormente, de nível de componentes para nível de *features*. Portanto, após essa análise, restaram as medidas de coesão EFD, IFD e LCC e as medidas de acoplamento DepIn, DepOut e SFC.

Nas seções seguintes, as medidas de software são apresentadas. Para exemplificar o cálculo das medidas, o exemplo de projeto hipotético de software de uma LPS (Figura 5.1) é mensurado utilizando as medidas selecionadas.

Figura 5.1 - Diagrama de Pacotes do Sistema de Software Hipotético



5.2.1 External-ratio Feature Dependency (EFD)

Nessa medida, é calculada a razão entre quantidade de dependências internas e a quantidade total de dependências (internas e externas) de uma determinada *feature*, definida pela Equação 5.1 (APEL; BEYER, 2011)

$$EFD(F) = \frac{|depInternas(F)|}{|depTotal(F)|} \tag{5.1}$$

sendo `depTotal(F)` o somatório das dependências internas `depInternas(F)` e das dependências externas `depExternas(F)` de uma *feature* `F`. Desse modo, caso uma *feature* `F` dependa apenas de componentes externos, então `EFD(F) = 0`, ou seja, `F` não é coesa. De outra forma, caso a *feature* `F` dependa apenas de componentes internos, então `EFD(F) = 1`, ou seja, a *feature* `F` é coesa. Ao aplicar a medida no projeto da LPS da Figura 5.1, tem-se que a `Feature_A` é "bastante" coesa (Equação 5.2), a `Feature_B` é coesa (Equação 5.3) e a `Feature_C` não é coesa (Equação 5.4).

$$EFD(Feature\_A) = \frac{|\{Componente\_B1, Componente\_A2\}|}{|\{Componente\_A1, Componente\_A2, Componente\_B1\}|} = \frac{2}{3} \tag{5.2}$$

$$EFD(Feature\_B) = \frac{|\{Componente\_B1\}|}{|\{Componente\_B1\}|} = \frac{1}{1} = 1 \tag{5.3}$$

$$EFD(Feature\_C) = \frac{|\emptyset|}{|\{Componente\_A2, Componente\_B1\}|} = \frac{0}{2} = 0 \tag{5.4}$$

5.2.2 Internal-ratio Feature Dependency (IFD)

Nessa medida, é calculada a razão entre a quantidade de dependências internas pela quantidade total de possíveis dependências internas de uma *feature*, definida pela Equação 5.5 (APEL; BEYER, 2011)

$$IFD(F) = \frac{|depInternas(F)|}{|componentes(F)|^2} \tag{5.5}$$

sendo `|depInternas(F)|` as dependências internas de uma *feature* `F` e `|componentes(F)|`^{2} o máximo de possíveis dependências da *feature* `F`. Desse modo, os componentes de uma *feature* dependem uns dos outros, incluindo auto-dependências, então `IFD(F) = 1`, ou seja, `F` é coesa. De modo contrário, caso os componentes da *feature* `F` não dependam uns dos outros, então `IFD(F) = 0`, ou seja, a *feature* `F` não é coesa. Ao aplicar a medida no projeto da LPS da Figura 5.1, tem-se que, a `Feature_A` é "moderadamente" coesa (Equação 5.6), a `Feature_B` é coesa (Equação 5.7) e a `Feature_C` não é coesa (Equação 5.8).

$$IFD(Feature\_A) = \frac{|\{Componente\_A1, Componente\_A2\}|}{|\{Componente\_A1, Componente\_A2\}|^2} = \frac{2}{2^2} = \frac{2}{4} = 0{,}5 \tag{5.6}$$

$$IFD(Feature\_B) = \frac{|\{Componente\_B1\}|}{|\{Componente\_B1\}|^2} = \frac{1}{1^2} = \frac{1}{1} = 1 \tag{5.7}$$

$$IFD(Feature\_C) = \frac{|\emptyset|}{|\{Componente\_C1, Componente\_C2\}|^2} = \frac{0}{2^2} = \frac{0}{4} = 0 \tag{5.8}$$

5.2.3 Lack of Concern-based Cohesion (LCC)

Nessa medida, é calculada a quantidade de *features* implementadas por um componente, ou seja, a *feature* em que o componente está contido é considerada no cálculo. Inicialmente, a medida é calculada em nível de componente, definida pela Equação 5.9 (SANTOS et al., 2017)

$$LCC(c) = |feature(o) \cup feature(a)| \tag{5.9}$$

sendo `feature(o)` uma *feature* atribuída ao componente `c` por meio da chamada/execução de operações e `feature(a)` uma *feature* atribuída ao componente por meio da declaração de atributos, passagem de parâmetros e instanciação de objetos do tipo do componente `c`. De modo a calcular a medida em nível de *feature*, é verificado se os componentes de uma *feature* `F` possuem *features* externas atribuídas a eles, definida pela Equação 5.10.

$$LCC_f(F) = \left| \bigcup_{i=0}^{n} LCC(i) \right| \quad (5.10)$$

sendo que, para uma *feature* `F`, contendo `n` componentes `c`, é verificado para cada componente da *feature* `ci` se possui alguma *feature* externa atribuída a ele. Desse modo, uma *feature* que não possui componentes implementando *features* externas é coesa, ou seja, `LCC(F) = 1`. De modo contrário, quanto maior a quantidade de *features* externas atribuídas aos componentes da *feature* `F`, menos coesa é a *feature* `F`. Ao aplicar a medida no exemplo da Figura 5.1, tem-se que a *feature* `Feature_A` é "moderadamente" coesa (Equação 5.11), a `Feature_B` é a mais coesa (Equação 5.12) e a `Feature_C` é a menos coesa (Equação 5.13).

$$LCC(Feature\_A) = |\{Feature\_A, Feature\_B\}| = 2 \quad (5.11)$$

$$LCC(Feature\_B) = |\{Feature\_B\}| = 1 \quad (5.12)$$

$$LCC(Feature\_C) = |\{Feature\_C, Feature\_B, Feature\_A\}| = 3 \quad (5.13)$$

5.2.4 Dependency In (DepIn)

Nessa medida, é calculada a quantidade de dependências em que a *feature* é o fornecedor (COLANZI et al., 2014). Originalmente, a medida é aplicada em meta-modelos, em que diferentes paradigmas de programação demandam uma estrutura de pacotes/diretórios para representação das features. A medida pode ser definida pela Equação 5.14.

$$DepIn(F) = \left| \bigcup_{i=0, i \neq F}^{n} feature(depExternas) \right| \quad (5.14)$$

sendo que as `n` *features* de um projeto LPS são percorridas para verificar se suas dependências externas `feature(depExternas)` são relacionadas a *feature* `F`. Desse modo, caso nenhuma *feature* dependa da *feature* `F`, então `DepIn(F) = 0`, ou seja, a *feature* `F` é desacoplada. De modo contrário, quanto mais *features* dependerem da *feature* `F`, maior será seu acoplamento. Ao aplicar a medida no exemplo da Figura 5.1, tem-se que a `Feature_A` é "moderadamente" acoplada (Equação 5.15), a `Feature_B` é a mais acoplada (Equação 5.16) e a `Feature_C` é a menos acoplada (Equação 5.17).

$$DepIn(Feature\_A) = |\{Feature\_C\}| = 1 \quad (5.15)$$

$$DepIn(Feature\_B) = |\{Feature\_A, Feature\_C\}| = 2 \quad (5.16)$$

$$DepIn(Feature\_C) = |\emptyset| = 0 \quad (5.17)$$

5.2.5 Dependency Out (DepOut)

Nessa medida, é calculada a quantidade de dependências em que a *feature* é o cliente (COLANZI et al., 2014). Da mesma forma que a medida DepIn, essa medida é aplicada em meta-modelos. A medida pode ser definida pela Equação 5.18.

$$DepOut(F) = \left| \bigcup_{i=0, i \neq F}^{n} feature(componentes) \right| \quad (5.18)$$

sendo que as `n` *features* de um projeto LPS são percorridas para verificar se seus componentes `feature(componentes)` são chamados pela *feature* `F`. Desse modo, caso a *feature* `F` não dependa de componentes de *features* externas, então `DepOut(F) = 0`, ou seja, a *feature* `F` é desacoplada. De modo contrário, quanto mais a *feature* `F` depender de *features* externas, maior será seu acoplamento. Ao aplicar a medida no exemplo da Figura 5.1, tem-se que a `Feature_A` é moderadamente acoplada (Equação 5.19), a `Feature_B` é a menos acoplada (Equação 5.20) e a `Feature_C` é a mais acoplada (Equação 5.21).

$$DepOut(Feature\_A) = |\{Feature\_\text{B}\}| = 1 \quad (5.19)$$

$$DepOut(Feature\_B) = |\emptyset| = 0 \quad (5.20)$$

$$DepOut(Feature\_C) = |\{Feature\_A, Feature\_B\}| = 2 \quad (5.21)$$

5.2.6 Structural Feature Coupling (SFC)

Nessa medida, é calculada a razão entre a quantidade de métodos compartilhados por duas features e a quantidade total de métodos associados as duas features (REVELLE et al., 2011). Como a medida calcula o acoplamento entre duas features, ela foi modificada de modo a calcular o acoplamento de uma *feature* com o restante das features, definida pela Equação 5.22.

$$SFC_f(F_a) = \frac{|M_a \cap (\bigcup_{i=1}^{n} M_i)|}{|\bigcup_{j=1}^{n} M_j|}, i \neq j \quad (5.22)$$

sendo $\bigcup_{i=1}^{n} M_i$ o conjunto de métodos de todas as *features* da LPS, com exceção do conjunto de métodos M_a, pertencente a *feature* analisada F_a. São utilizadas as propriedades distributivas para fazer a intersecção do conjunto de métodos da *feature* F_a com cada conjunto de métodos das *features* restantes $\bigcup_{i=1}^{n} M_i$. Posteriormente, é feita a união de todas as operações da LPS $\bigcup_{j=1}^{n} M_j$. Desse modo, quando as *features* não compartilham suas operações, então $SFC_f(F_a) = 0$, ou seja, a *feature* F_a está desacoplada. De modo contrário, caso as *features* compartilhem todos seus métodos, então $SFC_f(F_a) = 1$, ou seja, a *feature* F_a está acoplada. Ao aplicar a medida no projeto da LPS da Figura 5.1, têm-se que as operações pertencentes a uma *feature* são aquelas implementadas em componentes da *feature* ou chamadas de forma direta de outros componentes, o que indica que as *features* `Feature_A` e `Feature_C` são mais acopladas que a *feature* `Feature_B`. Dessa forma, o cálculo consiste em identificar as operações da `Feature_A` (Equação 5.23), da `Feature_B` (Equação 5.24), e da `Feature_C` (Equação 5.25).

$$M(Feature\_A) = \{Operacao\_A1, Operacao\_A2, Operacao\_B1\} \quad (5.23)$$

$$M(Feature\_B) = \{Operacao\_B1\} \quad (5.24)$$

$$M(Feature\_C) = \{Operacao\_A2, Operacao\_B1, Operacao\_C1, Operacao\_C2\} \quad (5.25)$$

A partir dessas operações, são feitas as intersecções e a união entre os métodos das *features*. De modo a facilitar o cálculo da medida SFC_f, as operações necessárias são abstraídas de modo a calcular a união das operações de todas as *features* (Equação 5.26) e pela intersecção entre cada par de *features*, isto é, `Feature_A` e `Feature_B` (Equação 5.27), `Feature_A` e `Feature_C` (Equação 5.28) e `Feature_B` e `Feature_C` (Equação 5.29).

$$Total = \left|\bigcup_{j=1}^{n} M_j\right|$$
$$Total = \{Operacao\_A1, Operacao\_A2, Operacao\_B1, Operacao\_C1, Operacao\_C2\} = 5 \quad (5.26)$$

$$A = Feature\_A \cap Feature\_B = \{Operacao\_B1\} \quad (5.27)$$

$$B = Feature\_A \cap Feature\_C = \{Operacao\_B1, Operacao\_A2\} \quad (5.28)$$

$$C = Feature\_B \cap Feature\_C = \{Operacao\_B1\} \quad (5.29)$$

resultando no cálculo da medida SFC_f, em que a `Feature_A` e a `Feature_C` são mais acopladas (Equação 5.30 e Equação 5.32) que a `Feature_B` (Equação5.31).

$$SFC_f(Feature\_A) = \frac{|A \cup B|}{Total} = \frac{2}{5} = 0{,}4 \quad (5.30)$$

$$SFC_f(Feature\_B) = \frac{|A \cup C|}{Total} = \frac{1}{5} = 0{,}2 \quad (5.31)$$

$$SFC_f(Feature\_C) = \frac{|B \cup C|}{Total} = \frac{2}{5} = 0{,}4 \quad (5.32)$$

5.3 Visão Geral

A ferramenta computacional (SPLiME) foi desenvolvida para automatizar o processo de coleta de medidas de software em LPS desenvolvida com as linguagens Jakarta e/ou AspectJ. Assim, espera-se amenizar possíveis erros quanto à forma de coleta de dados e de cálculo das medidas. O funcionamento da SPLiME é organizado em cinco processos (Figura 5.2): i) Entrada de Dados, são selecionadas as três LPS desenvolvidas com OA, com OC e com MCA; ii) Pré-processamento, o código fonte escrito na linguagem Jakarta é tratado; iii) Análise de Código, em que é feita a análise estática do código para encontrar dependências de código em nível de classe e, consequentemente, de feature; iv) Cálculo de Medidas, são aplicadas as regras para realizar o cálculo das medidas selecionadas (EFD, IFD, LCC, DepIn, DepOut, SFC, NOO, LOC, NOA e NOO); e v) Visualização de Medidas, o valor das medidas de cada LPS é apresentado em uma view do SPLiME e salvos em arquivos de extensão .csv.

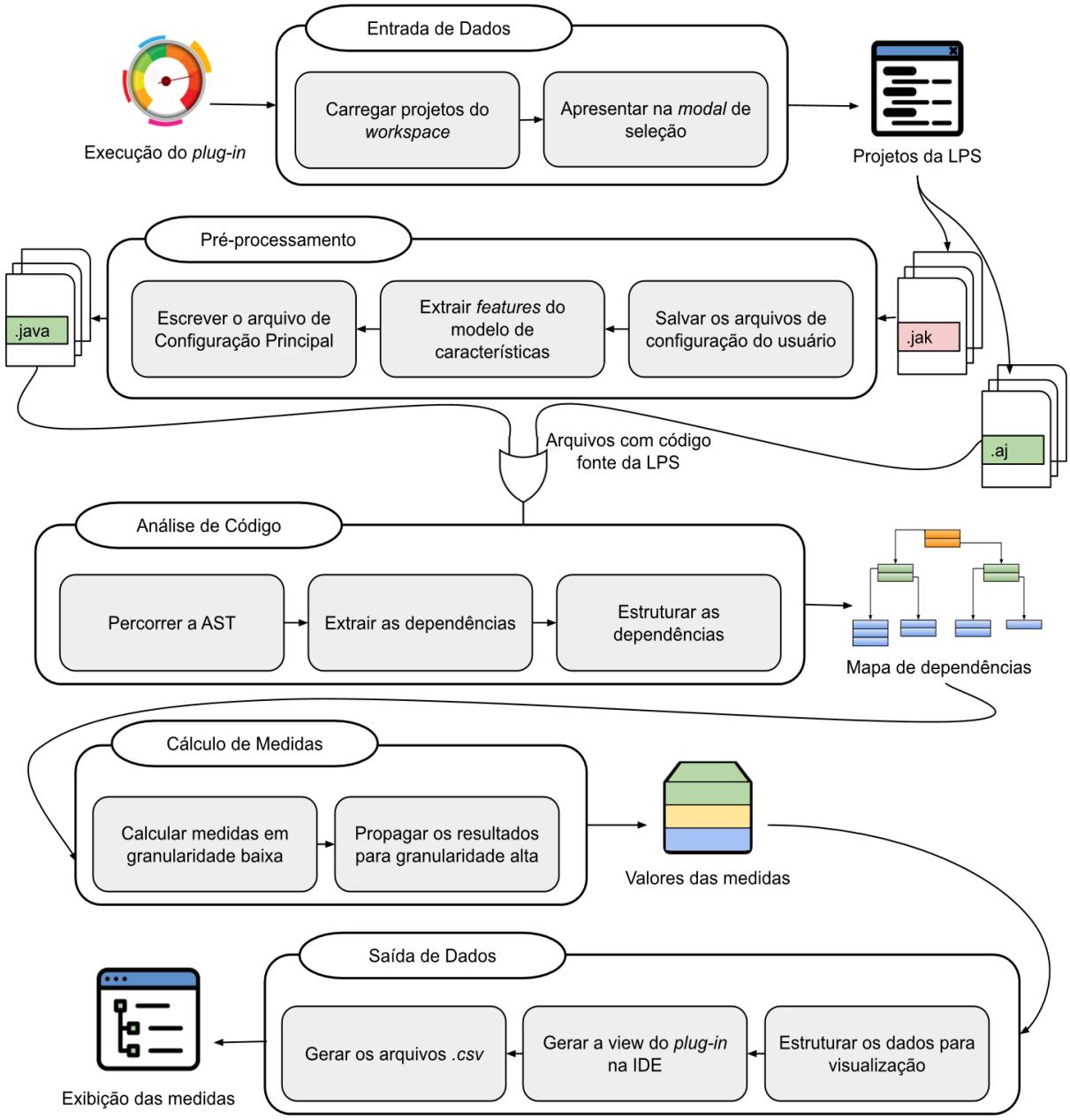
O usuário interage com o SPLiME por meio do processo Entrada de Dados e do processo Saída de Dados. Dessa forma, a interface do Eclipse IDE foi estendida pelo SPLiME para apresentar os elementos de interface (Figura 5.3): i) o workspace do Eclipse IDE, utilizado para seleção de projetos de entrada; ii) o ícone para a execução do SPLiME, inserido no menu suspenso da IDE; iii) a modal de seleção para a identificação dos projetos referentes a mesma LPS que o usuário deseja mensurar; iv) a view do SPLiME, que mostra as medidas calculadas em cada projeto em diferentes granularidades; e v) o botão de exportar o valor das medidas apresentadas na view como arquivos .csv.

Por exemplo, na Figura 5.3, o valor das medidas da LPS `HelloWorld`[17] é calculado. Para isso, os projetos implementados com as tecnologias são importados no *workspace* e selecionados na modal (passos i - iii). Em seguida, o valor das medidas é apresentado na *view*. Em nível de projeto, a medida LOC apresentou os valores 36, 18 e 50 linhas para as tecnologias OC, OA e MCA, respectivamente. Ao clicar sobre o nome de uma medida (*e.g.*, *Lines of Code*

[17] O exemplo está disponível de forma visual em: <https://www.youtube.com/watch?v=VIDNQybQEBU>

(LOC)), em nível de LPS, é apresentado o valor para essa medida em nível de *feature*, em que a LPS apresenta 4 *features* (`Beautiful`, `Hello`, `Wonderful` e `World`).

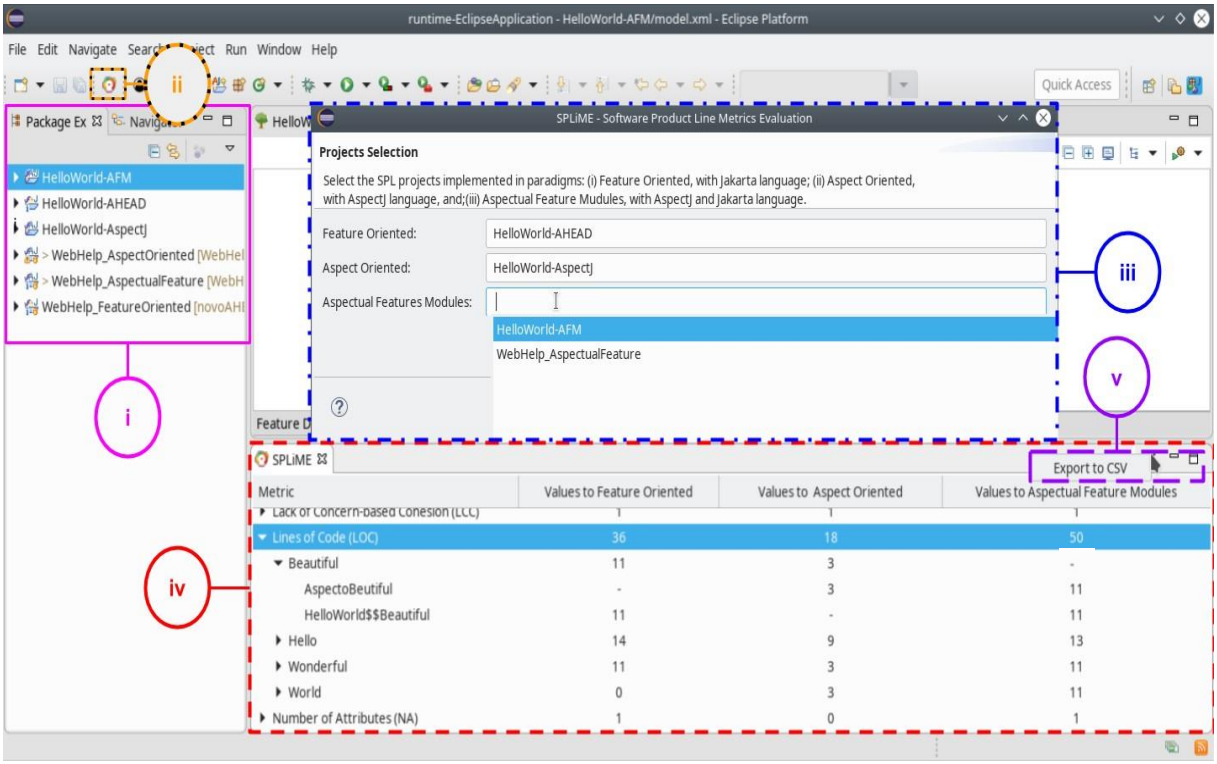
Figura 5.2 - Visão Geral da `SPLiME`



Para a *feature* `Beautiful`, a medida LOC apresentou os valores 11, 3 e 15 linhas para as tecnologias OC, OA e MCA, respectivamente. Ao clicar sobre o nome de uma *feature* (*e.g.*, `Beautiful`), é apresentado o valor da medida LOC em nível de componente. A *feature* `Beautiful` possui 2 componentes, o aspecto `AspectoBeautiful` e o refinamento de classe `HelloWorld$$Beautiful`. O aspecto `AspectoBeautiful` foi implementado apenas na tecnologia OA e possui 3 linhas e o refinamento de classe

`HelloWorld$$Beautiful` foi implementado nas tecnologias OC e MCA e possui 11 linhas em cada tecnologia. O valor das medidas em nível de *feature* pode ser exportado para `.csv` (passos iv e v).

Figura 5.3 - Extensão da Interface do Eclipse IDE



5.4 Arquitetura e Tecnologias Utilizadas

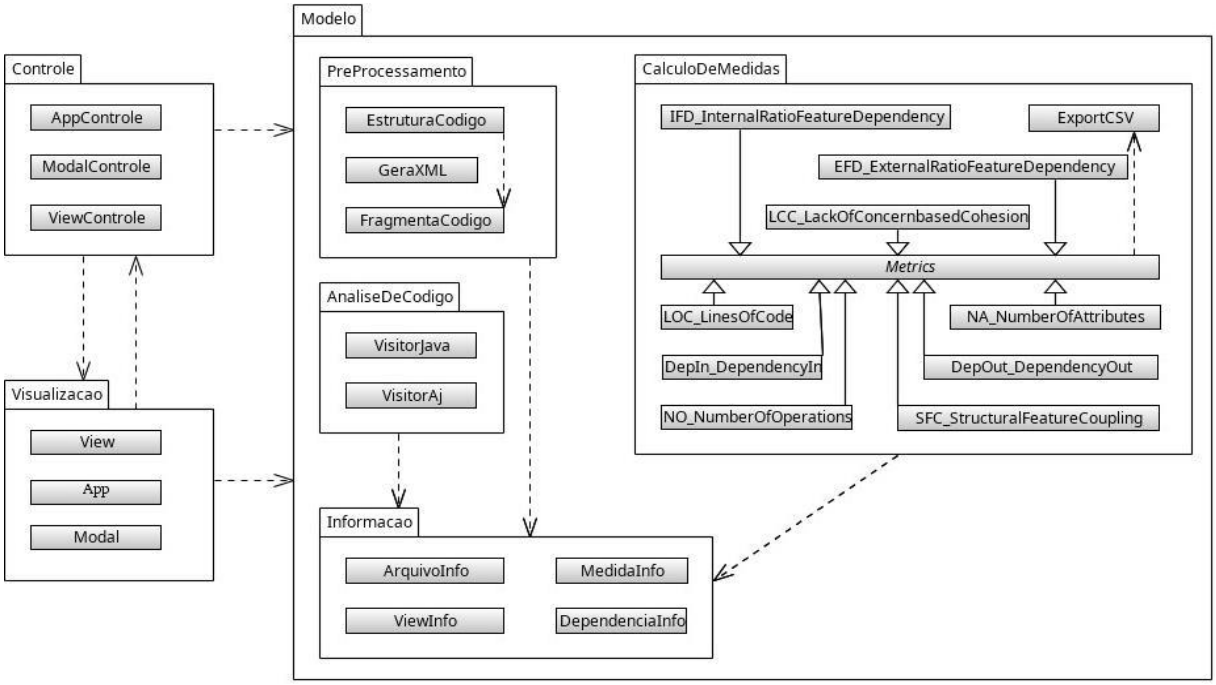
A arquitetura do `SPLiME` segue o padrão MVC (*Model-View-Controller*), que consiste em três principais componentes (DEACON, 2009): i) **Visualização**, renderiza o conteúdo do modelo a ser apresentado ao usuário e encaminha suas ações ao controlador; ii) **Modelo**, armazena, manipula e gera dados, podendo ser subdividida em modelo de domínio e modelo de aplicação; e iii) **Controle**, interpreta as ações do usuário e as mapeia para chamadas do modelo. Tal arquitetura é ilustrada pelo diagrama de pacotes na Figura 5.4, cujos componentes do MVC são caracterizados como pacotes na arquitetura da `SPLiME`[18].

No pacote `Visualização`, são apresentados os elementos da interface contidos nos processos **Entrada de Dados** e **Saída de Dados**. Na classe `App`, é realizada a extensão do Eclipse IDE adicionando o botão para execução do `SPLiME` e disponibilizando a *view* ao término do processamento dos dados. Na classe `Modal`, é apresentada a modal com os campos

[18] Código fonte da SPLiME disponível em: <https://github.com/LuanaAlmeidaMartins/Plugin-Medidas.git>

para seleção dos projetos da LPS. Na classe `View`, é configurada a *view* para a apresentação dos resultados do processamento e é adicionado o botão para exportar os resultados em arquivos `.csv`. Para a implementação dessas interfaces, foram utilizadas as bibliotecas `SWT/JFace`.

Figura 5.4 - Diagrama de Pacotes da `SPLiME`



Em geral, as ações solicitadas na interface são observadas por *listeners* presentes no pacote `Controle`. Para cada classe no pacote `Visualização`, foram criadas classes controladoras no pacote `Controle`, o que caracteriza a comunicação entre esses componentes do tipo um-para-um. Desse modo, ao identificar uma solicitação de ação na interface, são feitas chamadas de métodos das classes no pacote `Modelo` para o armazenamento de dados e a realização das operações para responder a ação solicitada. Com isso, tem-se que a comunicação entre os pacotes `Controle` e `Modelo` é do tipo um-para-muitos.

Para o armazenamento e a realização de operações, o pacote `Modelo` é organizado em: i) **modelo de domínio**, representado pelo pacote `Informacao`, onde as classes são independentes de classes externas ao pacote e expressam os objetos do domínio; e ii) **modelo de aplicação**, representado pelos pacotes `PreProcessamento`, `AnaliseDeCodigo`, e `CalculoDeMedidas`, realiza as operações necessárias em cada etapa de processamento. A comunicação no pacote `Modelo` é feita pelo acesso de classes/pacotes de domínio por classes/pacotes de aplicação. Além disso, esse pacote não se comunica com os outros pacotes, apenas o contrário. Especificamente, esse componente pode realizar chamadas de classes de aplicação e de classes de domínio para processar, estruturar e armazenar dados, enquanto que

o pacote `Visualização` pode acessar apenas classes de domínio para obter os dados a serem apresentados na interface.

Por exemplo, para iniciar o `SPLiME`, na classe `AppControle`, há um *listener* que identifica a chamada do botão de execução e habilita a *view* ao término do processamento dos dados (a classe `AppControle` chama a classe `View`). Na classe `View`, são construídos elementos de interface, correspondendo à tabela expansível (*TreeView*) e ao botão para exportar os resultados. Para exibir os dados na *TreeView*, nessa classe, há uma chamada direta para a classe `ViewInfo` do pacote `Modelo`, por meio da qual são obtidas as informações a serem exibidas na *view*. A ação de exibir os resultados foi desencadeada pelo *listener* que indicou a execução do `SPLiME`; portanto, o processamento e o armazenamento dos dados foram realizados, cabendo a classe `View` buscar as informações em classes de modelo de domínio. Diferentemente, o botão de exportar é "observado" por um *listener* da classe `ViewControle`, que chama a classe `ExportarCSV`, ao identificar que os resultados devem ser exportados.

5.5 Pré-processamento

O ponto principal para o desenvolvimento da `SPLiME` consiste na análise do código desenvolvido na linguagem Java, base para as linguagens AspectJ e Jakarta. A análise do código em Java é realizada por meio da árvore sintática abstrata (AST [19]), uma biblioteca disponibilizada pelo `Eclipse Java Development Tools` (JDT) que permite o acesso e a manipulação de código Java. Como essa biblioteca utiliza apenas códigos fonte Java, a LPS desenvolvida em Jakarta torna-se dependente da configuração de produto. Isto é, o código em Java é composto a partir da combinação de fragmentos de códigos fonte em Jakarta referentes a cada *feature* selecionada no modelo de características. Desse modo, ao utilizar a biblioteca AST, apenas uma configuração de produto é analisada por vez, o que não satisfaz o objetivo do trabalho: analisar o código da LPS quanto as propriedades de acoplamento e de coesão. Além disso, para a LPS em AspectJ, a biblioteca não identifica os adendos, pontos de corte e aspectos, prejudicando a análise do código.

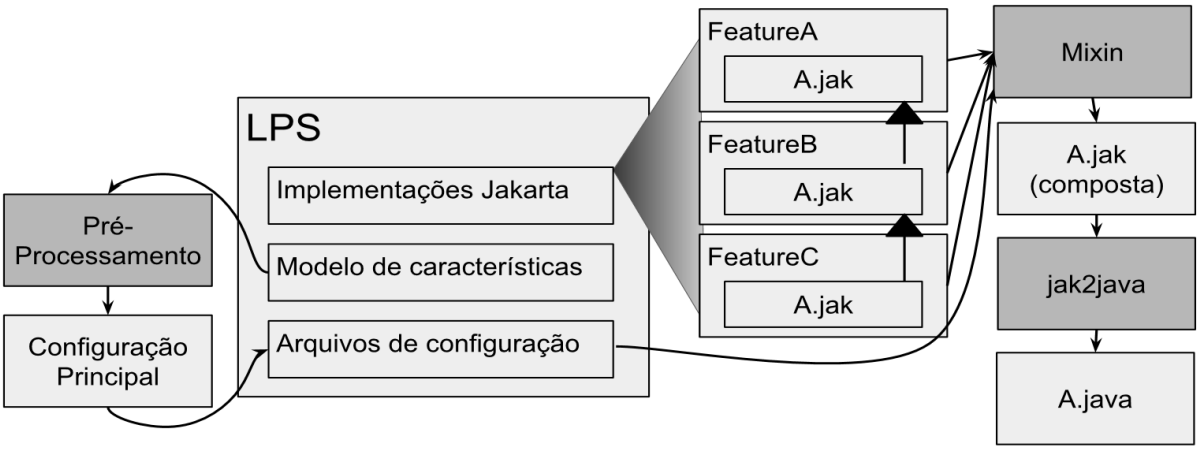
A princípio foram buscadas bibliotecas que poderiam substituir ou complementar a biblioteca Java para a criação e manipulação da AST. No entanto, foi encontrada uma biblioteca auxiliar apenas para a linguagem AspectJ, denominada AjAST [20]. Por meio da classe `AjASTVisitor`, essa biblioteca estende os métodos da classe `ASTVisitor` e adiciona os

[19] https://bit.ly/2Y9Ec8f
[20] https://bit.ly/2ULXH4Q

métodos necessários para a identificação de adendos, pontos de corte e aspectos. Em relação a linguagem Jakarta, não foram encontradas bibliotecas que deem suporte à criação e à manipulação de sua AST. No entanto, foram encontradas soluções na literatura (REIS et al., 2014), que sugeriam a substituição de palavras-chave da linguagem Jakarta por palavras-chave da linguagem Java (como substituir `refines` por `class`). No entanto, essa substituição não é o ideal, pois pode perder parte da sintaxe ao não considerar a composição do código. Dessa forma, o processamento proposto para `SPLiME` utiliza o arquivo de configuração da LPS, deixando que a composição e a transformação do código Jakarta em Java ocorram por meio do conjunto de ferramentas AHEAD. Na Figura 5.5, é apresentada o pré-processamento proposto para analisar o código da LPS em Java.

No processamento da linguagem Jakarta, são utilizadas duas principais ferramentas do AHEAD (`Mixin` e `jak2java`). `Mixin` é responsável por comprimir cadeias de refinamento em uma única interface ou classe. As cadeias de refinamento são geradas a partir da seleção de *features* presentes no modelo de características representada por um modelo de configuração. Dessa forma, `Mixin` recebe como entrada a configuração de um produto da LPS e as implementações `.jak`. Como resultado, são gerados arquivos `.jak` com as composições de classes. A partir das composições de classes, o arquivo `.java` é derivado por meio da ferramenta `jak2java`.

Figura 5.5 - Processamento da Linguagem Jakarta (Adaptado de Batory et al., (2004))



Como as cadeias de refinamento são geradas com base em um produto específico, a ideia do pré-processamento proposto é selecionar todas as *features* do modelo de características para que o produto final contenha cada refinamento de código da LPS. Dessa forma, o **Pré-Processamento** consiste em percorrer o modelo de características, identificando suas *features* para criar o arquivo **Configuração Principal** com todas as *features* selecionadas. Esse arquivo de configuração é temporário, não sendo permitido ao usuário utilizar o produto final gerado.

Inclusive, com esse arquivo de configuração, não é gerado um produto de software "compilável", visto que as restrições do modelo de características não são respeitadas.

No entanto, para análise estática, não é significativo o arquivo Configuração Principal apresentar uma configuração inválida, pois a composição do código ocorre de forma *bottom-up*. Isto é, a sequência de composição começa por uma *feature* filha estendendo sua *feature* pai, e assim por diante. Por exemplo, no sistema de `E-shop` (Figura 3.9), há uma restrição alternativa entre os meios de envio (`Shipment`), em que pode ser selecionado apenas um dos meios de envio, físico (`Physical Shipment`) ou eletrônico (`E-Shipment`). Na implementação das *features*, as *features* filhas (`Physical Shipment` e `E-Shipment`) seriam refinamentos da *feature* pai (`Shipment`). Como a composição começa das *features* filhas para as *features* pai, não haveria problema em ambos os refinamentos estarem presentes no código, pois o código seria `Physical Shipment extends Shipment` e `E-Shop extends Shipment`, respectivamente.

Para a criação do arquivo **Configuração Principal**, o modelo de características é percorrido em busca do nome das *features* da LPS. Posteriormente, um arquivo XML é criado com todas as *features* marcadas como selecionadas, independente das restrições impostas no modelo de características. No Código 5.1, é apresentado um arquivo XML seguindo as restrições do modelo de características, em que apenas uma das *features* `Physical Shipment` e `E-Shipment` pode ser selecionada (linhas 6 e 7). No Código 5.2, é apresentado o novo arquivo de configuração com todas as *features* automaticamente selecionadas.

Portanto, ao criar um novo arquivo de configuração com todas as *features*, a composição do `Mixin` comprime as cadeias de refinamento presentes nas *features* para, posteriormente, gerar o código na linguagem Java. Como resultado da compressão das cadeias de refinamento, as classes em Java possuem trechos de diversas *features* e, como parte do **Pré-Processamento**, trechos de uma classe foram separados e atribuídos às suas *features* de origem de modo a não perder informações sobre elas.

Código 5.1 - Arquivo de Configuração XML com as Restrições do Modelo de Características

```
1 <!------- Arquivo com restricoes ----->
2 <configuration>
3     <feature automatic= "selected" name= "EShop"/>
4     (...)
5     <feature automatic= "selected" name= "Shipment"/>
6     <feature manual= "selected" name= "PhysicalShipment"/>
7     <feature automatic= "unselected" name= "EShipment"/>
8 </configuration>
```

Código 5.2 - Arquivo de Configuração XML sem as Restrições do Modelo de Características

```
1 <!------- Arquivo sem restricoes ----->
2 <configuration>
3     <feature automatic= "selected" name= "EShop"/>
4     (...)
5     <feature automatic= "selected" name= "Shipment"/>
6     <feature automatic= "selected" name="PhysicalShipment"/>
7     <feature automatic= "selected" name= "EShipment"/>
8 </configuration>
```

5.6 Análise Sintática do Código da LPS

Para obter informações sobre o código, são utilizados `ASTVisitor` para Java e `AjASTVisitor` para AspectJ para adicionar um *visitor* a um elemento específico do código. Para os códigos fonte em Java, a classe `VisitorJava` estende a classe `ASTVisitor`, onde os seguintes *visitors* são sobrescritos:

- **`ClassInstanceCreation`**. É utilizado para coletar informações referentes à instanciação de objetos dentro de uma classe;
- **`FieldDeclaration`**. É utilizado para coletar informações referentes aos atributos declarados em uma classe;
- **`IResource`**. É utilizado para coletar informações referentes à classe, à quantidade de linhas e ao nome da classe;
- **`MethodDeclaration`**. É utilizado para coletar informações referentes ao método, como o nome, as variáveis/atributos utilizados e os parâmetros recebidos;
- **`MethodInvocation`**. É utilizado para coletar informações referentes as chamadas de métodos em uma classe e/ou método;
- **`PackageDeclaration`**. É utilizado para coletar informações referentes ao pacote, como o nome do pacote e as classes pertencentes a ele;
- **`TypeDeclaration`**. É utilizado para coletar informações referentes às extensões de classes e às implementações de interfaces.

Além disso, esses *visitors* foram complementados para a análise do código em Aspectj. Para isso, a classe `VisitorAj` estende a classe `AjASTVisitor`, onde os seguintes *visitors* são sobrescritos:

- **`BeforeAdviceDeclaration`**. É utilizado para coletar informações referentes aos adendos executados antes de um ponto de junção;
- **`AroundAdviceDeclaration`**. É utilizado para coletar informações referentes aos adendos executados com comportamento personalizado, responsáveis por decidir o fluxo de execução;
- **`AfterAdviceDeclaration`**. É utilizado para coletar informações referentes aos adendos executados após um ponto de junção. Como o adendo pode ser interceptado pelo retorno da execução (*After returning*) ou pela exceção do método (*After throwing*), são utilizados, respectivamente, os *visitors* `AfterReturningAdviceDeclaration` e `AfterThrowingAdviceDeclaration`.

A partir desses *visitors*, as dependências entre classes são armazenadas em uma estrutura definida como `DependenciaInfo`, que contém o atributo `IType`, por meio do qual é obtido o nome, a quantidade de linhas, os atributos e os métodos declarados na classe, e o atributo `ArrayList<String>`, em que as dependências da classe são armazenadas, considerando a passagem de parâmetros, atributos declarados, objetos instanciados e chamadas de métodos. Além disso, as *features* da LPS foram consideradas para a rastreabilidade entre as tecnologias utilizadas para sua implementação. Dessa forma, foi criado o atributo `HashMap<String, ArrayList<DependenciaInfo>>`, sendo a chave do *hash* o nome da *feature* e seu valor uma lista de classes com as dependências identificadas. Esse *hash* representa a estrutura da LPS, sendo utilizada para o cálculo de medidas.

5.7 Cálculo de Medidas

Na Tabela 5.1, são apresentadas as medidas calculadas pela `SPLiME`. Além do nome e da sigla, cada medida possui:

- **Propriedade**. É uma abstração que caracteriza um objeto. No `SPLiME`, as medidas contemplam três propriedades: i) tamanho; ii) coesão; e iii) acoplamento. No entanto, apenas as medidas de coesão e de acoplamento são utilizadas para mensurar a modularidade da LPS;
- **Granularidade**. É utilizada para definir o nível em que as medidas são aplicadas. No `SPLiME`, as medidas são calculadas em três níveis (Tabela 5.2): i) componentes; ii) *features*; e iii) sistema. Cabe ressaltar que o nível mais baixo são os componentes; assim, de modo a calcular as medidas nos níveis superiores, sendo o sistema o nível mais

alto, as medidas são propagadas. Essa propagação pode ser dada pela soma (*SUM*) ou pela média (*AVG*) das medidas do nível inferior. Por exemplo, a medida LOC é calculada em nível de componentes. Para o nível de *features*, os valores de cada componente são somados. Da mesma forma, para o nível de projeto, os valores de cada *feature* são somados;

- **Intervalo**. É referente ao valor retornado pela medida. Para medidas representadas pelo intervalo `[0, n]`, são retornados valores inteiros positivos. Para as medidas representadas pelo intervalo `[0, 1]`, são retornados valores racionais positivos.

Tabela 5.1 - Caracterização das Medidas Coletadas pela `SPLiME`

Medida	Sigla	Propriedade	Granularidade	Intervalo
Dependency In	DepIn	Acoplamento	Feature	[0, n]
Dependency Out	DepOut	Acoplamento	Feature	[0, n]
Extenal-ratio Feature Dependency	EFD	Coesão	Feature	[0, 1]
Internal-ratio Feature Dependency	IFD	Coesão	Feature	[0, 1]
Lack of Concern-based Cohesion	LCC	Coesão	Componente	[0, n]
Lines of Code	LOC	Tamanho	Componente	[0, n]
Number of Attributes	NOA	Tamanho	Componente	[0, n]
Number of Operations	NOO	Tamanho	Componente	[0, n]
Structural Feature Coupling	SFC	Acoplamento	Componente	[0, 1]

Tabela 5.2 - Elementos de Orientação a Aspectos e de Orientação a Características

Elementos	Orientação a Aspectos (AspectJ)	Orientação a Características (Jakarta)
Sistema	Sistema	Sistema, Extensões de sistema
Feature	Pacote	*Layer* (diretório)
Componente	Classe, aspecto e interface	Classe, refinamento de classe
Atributos	Campo, variável e *inter-type*	Campo e variável
Operações	Método, construtor, *inter-type* e *advice*	Método, construtor e expressão

A seguir, é apresentada a forma como as medidas do `SPLiME` foram calculadas. No Código 5.3, é apresentado o cálculo da medida LOC. No *loop* interno (linhas 4 a 7), o *array* `medidaComponente` é preenchido com o valor da medida LOC obtida por meio do método `getNumberOfLines` (linha 5) para cada componente da *feature* analisada. Posteriormente, o *array* `medidaComponente` é inserido em sua respectiva *feature* (linhas 8 e 9), para qual é indicado o tipo de propagação do cálculo da medida. Similarmente, as medidas NOO e NOA são calculadas de forma direta pela AST por meio da substituição do método `getNumberOfLines` por `getNumberOfOperations` e `getNumberOfAttributes`, respectivamente.

Código 5.3 - Lógica para as Medidas LOC, NOA e NOO

```
   Entrada: estrutura LPS, array medidaFeature
   Saída: array medidaFeature

 1 Início
 2   para cada (feature ∈ LPS) faça
 3     array medidaComponente <- inicializa
 4     para cada (componente ∈ feature) faça
 5       valor = MedidaInfo(componente.Nome,
           componente.getNumberOfLines)
 6       medidaComponente.adiciona(valor)
 7     fim-para-cada
 8     valor = MedidaInfo(feature.Nome, medidaComponente,
         Propaga(SUM))
 9     medidaFeature.adiciona(valor)
10   fim-para-cada
11   retorna medidaFeature
12 Fim
```

O cálculo da medida DepIn é apresentado no Código 5.4. Inicialmente, os componentes da *feature* analisada são listados por meio do método `getComponentes` e adicionados no *array* `featureComp` (linha 4). Posteriormente, as outras *features* da LPS, exceto a *feature* analisada (linhas 5 e 6), são percorridas de modo a identificar em suas dependências os componentes da *feature* analisada no *loop* interno (linhas 7 a 11). Caso o nome do componente da *feature* analisada seja identificado nas dependências de outras *features*, então a *feature* dependente é inserida no *array* `featureDependente`. Finalmente, no *loop* externo, a quantidade de *features* dependentes da *feature* analisada é inserida no *array* `medidaFeature` (linhas 14 e 15).

De forma contrária, para o cálculo da medida DepOut, são buscados componentes de outras *features* da LPS nas dependências da *feature* analisada. No Código 5.5, é apresentada a lógica utilizada para o cálculo dessa medida. Inicialmente, todas as dependências da *feature* analisada são adicionadas no *array* `featureDep` (linha 4), por meio do método `getDependencias`. Em seguida, cada uma das dependências contidas nesse *array* é comparada com o nome dos componentes presentes no restante das *features* da LPS (linhas 5 a 11). Ao encontrar o nome de um componente igual ao nome de uma dependência, então essa *feature* é adicionada no *array* `featureUsada`, que indica a quantidade de dependências de saída da *feature* analisada (linha 8). Similarmente, no cálculo da medida LCC, são buscados componentes pertencentes à *feature* analisada e às outras *features* da LPS nas dependências da *feature* analisada.

Código 5.4 - Lógica para a medida DepIn

```
   Entrada: estrutura LPS, array medidaFeature
   Saída: array medidaFeature

 1 Início
 2 |  para cada (featureAnalisada ∈ LPS) faça
 3 |  |  array featureDependente <- inicializa
 4 |  |  array featureComp <- getComponentes(featureAnalisada)
 5 |  |  para cada (feature ∈ LPS) faça
 6 |  |  |  se(feature.Nome != featureAnalisada.Nome) então
 7 |  |  |  |  para cada (componente ∈ feature) faça
 8 |  |  |  |  |  se(componente.Dependencia in featureComp)então
 9 |  |  |  |  |  |  featureDependente.adiciona(feature.Nome)
10 |  |  |  |  |  fim-se
11 |  |  |  |  fim-para-cada
12 |  |  |  fim-se
13 |  |  fim-para-cada
14 |  |  valor = MedidaInfo(featureAnalisada.Nome,
   |  |    featureDependente.Tamanho)
15 |  |  medidaFeature.adiciona(valor)
16 |  fim-para-cada
17 |  retorna medidaFeature
18 Fim
```

Código 5.5 - Lógica para a Medida DepOut

```
   Entrada: estrutura LPS, array medidaFeature
   Saída: array medidaFeature

 1 Início
 2 |  para cada (featureAnalisada ∈ LPS) faça
 3 |  |  array featureUsada <- inicializa
 4 |  |  array featureDep<-getDependencias(featureAnalisada)
 5 |  |  para cada (feature ∈ LPS) faça
 6 |  |  |  se(feature.Nome != featureAnalisada.Nome) então
 7 |  |  |  |  se(featureDep in feature.Dependencia)
 8 |  |  |  |  |  featureUsada.adiciona(feature.Nome)
 9 |  |  |  |  fim-se
10 |  |  |  fim-se
11 |  |  fim-para-cada
12 |  |  valor= MedidaInfo(featureAnalisada.Nome,
   |  |    featureUsada.Tamanho)
13 |  |  medidaFeature.adicona(valor)
14 |  fim-para-cada
15 |  retorna medidaFeature
16 Fim
```

O cálculo para a medida EFD é apresentado no Código 5.6. A princípio, todos os componentes da LPS são adicionados no *array* `todosComponentes`, por meio do método `getComponentesLPS` (linha 2). Adicionalmente, os componentes da *feature* analisada são adicionados no *array* `featureComponente`, por meio do método `getComponentes`

(linha 5). Dessa forma, no *loop* interno (linhas 6 a 13), as dependências de cada componente da *feature* analisada são classificadas como dependências internas e/ou dependências totais. Posteriormente, a quantidade total de dependências internas é dividida pela quantidade total de dependências da *feature* analisada, a fim de obter e adicionar o valor da medida EFD na *feature* analisada (linhas 14 e 15). Similarmente, para o cálculo da medida IFD, as dependências da *feature* analisada são classificadas como dependências internas e, posteriormente, a quantidade total de dependências internas é dividida pelo quadrado de quantidade de componentes da *feature* analisada.

Código 5.6 - Lógica para a Medida EFD e IFD

```
   Entrada: estrutura LPS, array medidaFeature
   Saída: array medidaFeature

 1 Início
 2     array todosComponentes = getComponentesLPS()
 3     para cada (feature ∈ LPS) faça
 4         int todasDep = 0, internasDep = 0
 5         array featureComponente <- getComponentes(feature)
 6         para cada (componente ∈ feature) faça
 7             se(componente.Dependencia in todosComponentes) então
 8                 todasDep++
 9             fim-se
10             se(componente.Dependencia in featureComponente)
               então
11                 internasDep++
12             fim-se
13         fim-para-cada
14         valor=MedidaInfo(feature.Nome, internasDep/todasDep)
15         medidaFeature.adicona(valor)
16    fim-para-cada
17    retorna medidaFeature
18 Fim
```

O cálculo da medida SFC (Código 5.7) consiste em fazer a intersecção dos métodos utilizados ou implementados na *feature* analisada com os métodos utilizados ou implementados em cada uma das *features* restantes. Além disso, é feita a união dos métodos utilizados ou implementados por todas as *features* da LPS. Desse modo, todos os métodos utilizados ou implementados na *feature* analisada são adicionados no *array* `featureMetodos`, por meio do método `getTodosMetodos` (linha 3) e são adicionados no *array* `uniao` (linha 5). Ainda, esse *array* é preenchido com todos os métodos de outras *features* da LPS (linhas 7 a 9). Posteriormente, todos métodos das *features* da LPS, exceto a *feature* analisada, são procurados no *array* `featureMetodos`; caso algum método seja encontrado, ele é adicionado no *array* `interseccao` (linhas 10 a 14). Finalmente, o valor da medida é calculado para a *feature*

analisada por meio da divisão entre o tamanho do *array* `interseccao` e o tamanho do *array* `uniao` (linhas 16 e 17).

Código 5.7 - Lógica para a Medida SFC

```
   Entrada: estrutura LPS, array medidaFeature
   Saída: array medidaFeature

 1 Início
 2     para cada (featureAnalisada ∈ LPS) faça
 3       array featureMetodos<-getTodosMetodos(featureAnalisada)
 4       array interseccao, uniao <- inicializa
 5       uniao.adicionaTodos(featureMetodos)
 6       para cada (feature ∈ LPS) faça
 7         se(feature.Nome != featureAnalisada.Nome) então
 8           uniao.adicionaTodos(getTodosMetodos(feature))
 9         fim-se
10         para cada(componente ∈ feature) faça
11           se(componente.Metodo in featureMetodos) então
12           interseccao.adiciona(componente.Metodo)
13           fim-se
14         fim-se
15       fim-para-cada
16       valor = MedidaInfo(feature.Nome,
           interseccao.Tamanho/uniao.Tamanho)
17       medidaFeature.adicona(valor)
18   fim-para-cada
19   retorna medidaFeature
20 Fim
```

É importante destacar que os valores das medidas obtidos por meio da utilização do SPLiME foram verificados via oráculo. Para isso, alguns projetos disponibilizados como exemplos pela FeatureIDE foram utilizados para a coleta manual e automática das medidas de software. Dessa forma, os valores das medidas obtidos com o SPLiME deveriam ser iguais aos valores coletados de forma manual.

5.8 Considerações Finais

A partir da identificação das propriedades da LPS a serem mensuradas, foram escolhidas medidas de software, considerando as mais referenciadas na literatura. Como resultado, foram selecionadas as medidas de acoplamento DepIn, DepOut, SFC e as medidas de coesão EFD, IFD e LCC. Essas medidas foram utilizadas para a mensuração da modularidade da LPS desenvolvida neste trabalho.

Para obtenção do valor de medidas de software, há diversas ferramentas, por exemplo, o Eclipse Metrics e o Analizo. No entanto, essas ferramentas são destinadas para sistemas de software desenvolvidos na linguagem Java. Como as linguagens AspectJ e Jakarta são

extensões da linguagem Java, aparentemente, tais ferramentas seriam capazes de coletar as medidas da LPS. Porém, como se tratam de paradigmas diferentes, as ferramentas não expressam características específicas da LPS. Por exemplo, com o Eclipse Metrics, é obtido o valor de medidas relacionadas a OO (medidas de QMOOD (BANSIYA; DAVIS, 2002) e CK (CHIDAMBER; KEMERER, 1994)). Essas medidas não são satisfatórias para identificar o entrelaçamento e o espalhamento de código entre as *features*. Além disso, ao analisar como essas ferramentas funcionam em sistemas desenvolvidos em Jakarta, apenas uma configuração de produto é analisada por vez, visto que a ferramenta depende da composição e tradução do código Jakarta. Quando aplicadas em sistemas de software desenvolvidos em AspectJ, não reconhecem a declaração de adendos, aspectos e outros.

De modo a contornar o problema de analisar apenas uma configuração de produto, foi proposta uma substituição (REIS et al., 2014) de palavras-chave das linguagens Jakarta e AspectJ por palavras-chave da linguagem Java (*e.g.*, as palavras-chave `refines` e `aspect` foram substituídas por `class`). No entanto, essa substituição não é ideal, pois há perda de sintaxe que pode influenciar na análise do código. Por exemplo, um refinamento de classe `.jak` é composto como uma extensão em uma classe `.java`. Ao substituir o `refines`, é criada uma nova classe, perdendo a informação da extensão de classe. Além disso, com essa ferramenta computacional, não são exploradas medidas específicas para LPS.

Diferentemente, `SPLiME` permite analisar o código da LPS desenvolvida em Jakarta com a criação de um arquivo de configuração com todas as *features* do modelo de características selecionadas. Apesar da configuração gerada ser inválida, pois as restrições do modelo de características não são satisfeitas, isso não interfere na composição do código. Além disso, foi encontrada uma biblioteca específica para a análise do código em AspectJ, eliminando quaisquer problemas relacionados a sintaxe. Adicionalmente, `SPLiME` coleta medidas de software relacionadas ao acoplamento e à coesão, medidas de software frequentemente utilizadas na literatura para mensurar a modularidade de LPS.

6 AVALIAÇÃO DA ABORDAGEM

6.1 Considerações Inicias

Para a avaliação da abordagem, foi escolhido o domínio de TA. Normalmente, ao utilizar recursos de TA, as pessoas com deficiência ou com dificuldades de aprendizagem específicas (*e.g.*, dislexia) enfrentam obstáculos como alto custo de aquisição e dificuldades em encontrar recursos que satisfaçam suas necessidades de forma satisfatória (CGEE, 2012). Dada a alta personalização de recursos exigida por essas pessoas, torna-se perceptível os desafios enfrentados pelos desenvolvedores na implementação de sistemas de software de TA. As habilidades funcionais variam de pessoa para pessoa, necessitando dos mais variados recursos desses sistemas, o que aumenta seu custo. Nessa perspectiva, a utilização de LPS para o desenvolvimento de recursos de TA faz-se importante, pois os artefatos de software são reutilizados, resultando em benefícios como a redução do custo de desenvolvimento e maior personalização dos sistemas de software.

De modo a investigar como LPS é utilizada para o desenvolvimento de recursos de TA, uma revisão *ad hoc* da literatura foi realizada (MARTINS et al., 2018). Como resultado, foi identificado que LPS ainda não foi amplamente explorada para esse desenvolvimento. Dentre os estudos encontrados, há pouca descrição de como as atividades do ciclo de vida de LPS podem ser realizadas, bem como os estudos não focaram na avaliação da qualidade da LPS desenvolvida. Diante disso, neste trabalho, a avaliação da abordagem é dada em termos do desenvolvimento de uma LPS de recursos de TA para pessoas com dislexia. Para isso, são contempladas as atividades da Engenharia de Domínio, que visam o desenvolvimento dos artefatos de software da LPS a serem reutilizados durante a derivação de produtos. Uma vez que a modularidade da LPS pode influenciar na reusabilidade dos artefatos de software, é realizada uma análise estatística de modo verificar se existe diferença significativa entre a LPS implementada com diferentes tecnologias de gerenciamento de variabilidades, em termos da coesão e do acoplamento entre suas *features*.

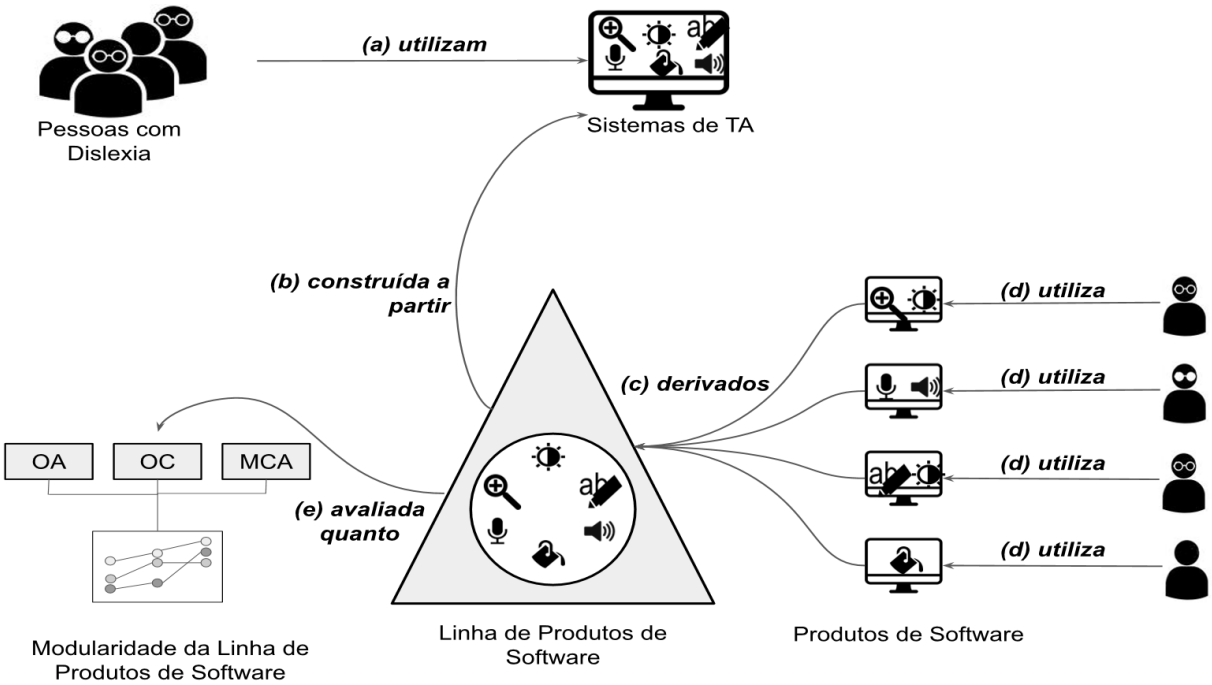
O restante deste capítulo está organizado da seguinte forma. Na Seção 6.2, é apresentado o desenho do estudo, em que é mostrado como LPS pode ser utilizada para o desenvolvimento de recursos de TA. Na Seção 6.3, é apresentada a análise de domínio, em que diversos sistemas de software de TA para pessoas com dislexia são analisados, a fim de identificar recursos comuns e variáveis entre esses sistemas. Na Seção 6.4, é apresentada a arquitetura de domínio, no qual são definidos as variantes e os pontos de variabilidade da LPS para a sua modelagem.

Na Seção 6.5, é apresentado o processo de implementação das *features* da LPS, que consistiu na utilização de três tecnologias de gerenciamento de variabilidades. Na Seção 6.6, é apresentado um exemplo de derivação de produtos, que representa a forma como foi verificada a corretude da implementação das *features*. Na Seção 6.7, é apresentada a análise estatística para verificar se há diferença significativa entre a LPS implementada com diferentes tecnologias de gerenciamento de variabilidades. Na Seção 6.8, é apresentada a discussão dos resultados obtidos na avaliação da abordagem.

6.2 Desenho do Estudo

Para pessoas com deficiência e com dificuldades de aprendizagem, encontrar sistemas de software que forneçam um conjunto de funções adequadas às suas necessidades é um fator limitante para a utilização de sistemas de software de TA. Muitas vezes, esses sistemas usados por essas pessoas fornecem funções não necessárias para seu contexto, o que aumenta o custo de aquisição e a carga cognitiva para a sua utilização. Dessa forma, a abordagem utilizada nesse trabalho consiste em desenvolver uma LPS a partir de sistemas de software de TA para pessoas com dislexia. Na Figura 6.1, é apresentado o desenho do estudo.

Figura 6.1 - Desenho de Estudo para o Desenvolvimento do Trabalho



A princípio, um grupo de pessoas com dislexia utiliza um ou mais sistemas de software de TA para acessar páginas Web (Figura 6.1(a)). Esses sistemas fornecem diversas funções relacionadas à navegação da página, à apresentação de texto, à organização de conteúdo e à

linguagem utilizada, de modo a atender dificuldades de aprendizagem específicas dessas pessoas (AVELAR et al., 2015). A LPS é construída a partir desses sistemas utilizando a abordagem extrativa, ou seja, as *features* dos sistemas são identificadas e extraídas para comporem a LPS. Para isso, é utilizado o processo de Engenharia de Domínio (Figura 6.1(b)). Durante esse processo, são realizadas as atividades:

- Análise de Domínio, em que diversos sistemas pertencentes ao domínio de TA para pessoas com dislexia foram analisados para identificar os recursos frequentemente disponibilizados;
- Arquitetura de Domínio, em que as *features* comuns e as *features* variáveis foram identificadas para a modelagem de características;
- Implementação do Domínio, em que a LPS foi desenvolvida utilizando três tecnologias de gerenciamento de variabilidades;
- Teste de Domínio, em que foram derivados produtos a fim de verificar a implementação das *features*.

Com a derivação de produtos, cada pessoa com dislexia utiliza um sistema de software que atende melhor as suas dificuldades de aprendizagem específicas (Figura 6.1(d)). Dessa forma, são fornecidas funções realmente necessárias, diminuindo a carga cognitiva à navegação em páginas Web e diminuindo o custo de aquisição de sistemas de software de TA. É importante destacar que as tecnologias de gerenciamento de variabilidades utilizadas para a implementação da LPS podem influenciar no reúso efetivo dos artefatos de software. Portanto, a escolha de uma tecnologia menos adequada para a modularização da LPS pode dificultar a adaptação desses sistemas conforme as necessidades dos usuários (Figura 6.1(c)). A LPS é avaliada quanto à modularidade das tecnologias de gerenciamento de variabilidades utilizadas para sua implementação (Figura 6.1(e)).

6.3 Análise de Domínio

Para o estudo, foi definido trabalhar com o domínio de sistemas de software para auxilio à leitura de textos por pessoas com dislexia. Além de investigar as tecnologias de gerenciamento de variabilidades, o estudo contribui para o estado da arte no sentido de apresentar uma LPS desenvolvida de modo a derivar recursos de TA. Desse modo, foram investigados sistemas de software e extensões para navegadores que facilitem a leitura, a escrita e a interpretação de textos por pessoas com dislexia. Ao todo, foram identificados 23 recursos (Tabela 6.1)

disponibilizados em 6 sistemas de software (ClaroRead, kurzweil 3000, Read&Write, Dolphin EasyReader, ATBar, MyStudyBar) e 4 extensões para navegadores (WebHelpDyslexia, DyslexiaReaderChrome, DyslexiaFormatter e VoiceRead).

Dentre esses recursos, foram selecionados os que estão presentes em, pelo menos, 5 sistemas/extensões para serem implementados na LPS. Desse modo, foram selecionados os recursos: tamanho, família e cor da fonte; espaçamento entre parágrafos, linhas e caracteres; mudança na cor de fundo (*Background*); adição de transparência (*Overlay*); régua de leitura; grifar o texto (*Highlight*); remover itálico, negrito e sublinhado; e converter texto para áudio. Essa seleção foi feita para identificar *features* comuns e *features* variáveis da LPS. Os recursos disponibilizados por quantidade menor de sistemas foram considerados não tão necessários para compor as *features* iniciais da LPS, podendo ser adicionados posteriormente. Além disso, é possível notar que, os recursos disponibilizados por mais sistemas, não há um recurso disponibilizado por todos os sistemas, o que permite que, dentre os recursos selecionados para LPS, exista bastante variabilidade.

Tabela 6.1 - Recursos Identificados na Análise de Domínio

Recursos / Sistemas/ Extensões	Tamanho da Fonte	Família da Fonte	Cor da Fonte	Espaçamento de Caracteres	Espaçamento de Linhas	Espaçamento de Parágrafos	Cor de Fundo	*Overlay*	Régua	*Highlight*	Negrito	Sublinhado	Alinhamento	Itálico	Dicionário	Texto para Áudio	Áudio para Texto	Preditor de Palavras	Verificador de Ortografia	Mapa Mental	OCR	Criação de Notas	Tradutor
Sistemas de Software																							
ClaroRead	X	X	X	X	X	X	X	X	X	X					X	X	X	X	X	X	X	X	
Kurzweil 3000	X	X	X				X			X					X	X	X	X		X		X	
Read&Write										X	X	X		X	X	X	X	X	X		X	X	X
Dolphin Easy Reader	X	X	X	X	X	X	X			X						X							
ATBar	X	X	X				X	X	X	X						X				X			
MyStudyBar	X	X	X				X	X	X	X					X	X	X	X		X			
Extensões Navegadores																							
WebHelpDyslexia	X	X	X	X	X	X	X	X	X	X	X	X	X	X		X							
DyslexiaReaderChrome	X	X	X	X	X	X	X																
VoiceRead	X	X	X	X	X		X			X						X							
DyslexiaFormatter	X	X	X	X	X	X	X	X	X							X							

6.4 Arquitetura de Domínio

Foram identificadas *features* comuns e *features* variáveis da LPS. Normalmente, alta porcentagem de *features* comuns exige menor esforço para projetar e implementar a LPS; por

outro lado, as *features* variáveis determinam a quantidade potencial de diferentes aplicações que podem ser derivadas da LPS, impactando sobre as metas e as necessidades dos usuários (APEL et al., 2013). Como observado na Análise Domínio, as habilidades funcionais de pessoas com dislexia variam bastante, o que demanda que os sistemas disponibilizem os mais diversos recursos. Desse modo, na construção do modelo de domínio, os recursos identificados na Análise de Domínio foram definidos como opcionais para permitir mais customizações dos produtos da LPS.

O modelo de domínio foi feito por meio de uma estrutura de árvore, para a qual foi utilizado o *plug-in* FeatureIDE[21] no Eclipse Oxygen (4.7.3)[22]. Esse *plug-in* foi escolhido por oferecer suporte gráfico e textual para o modelo de características, verificando de forma automatizada as anomalias que podem ocorrer durante a modelagem. Para a implementação de domínio, é oferecido suporte para diversas linguagens, dentre as quais encontram-se Jakarta (AHEAD) e AspectJ. Além disso, pode-se gerar de forma automatizada sistemas de software por meio da escolha de uma configuração de *features*.

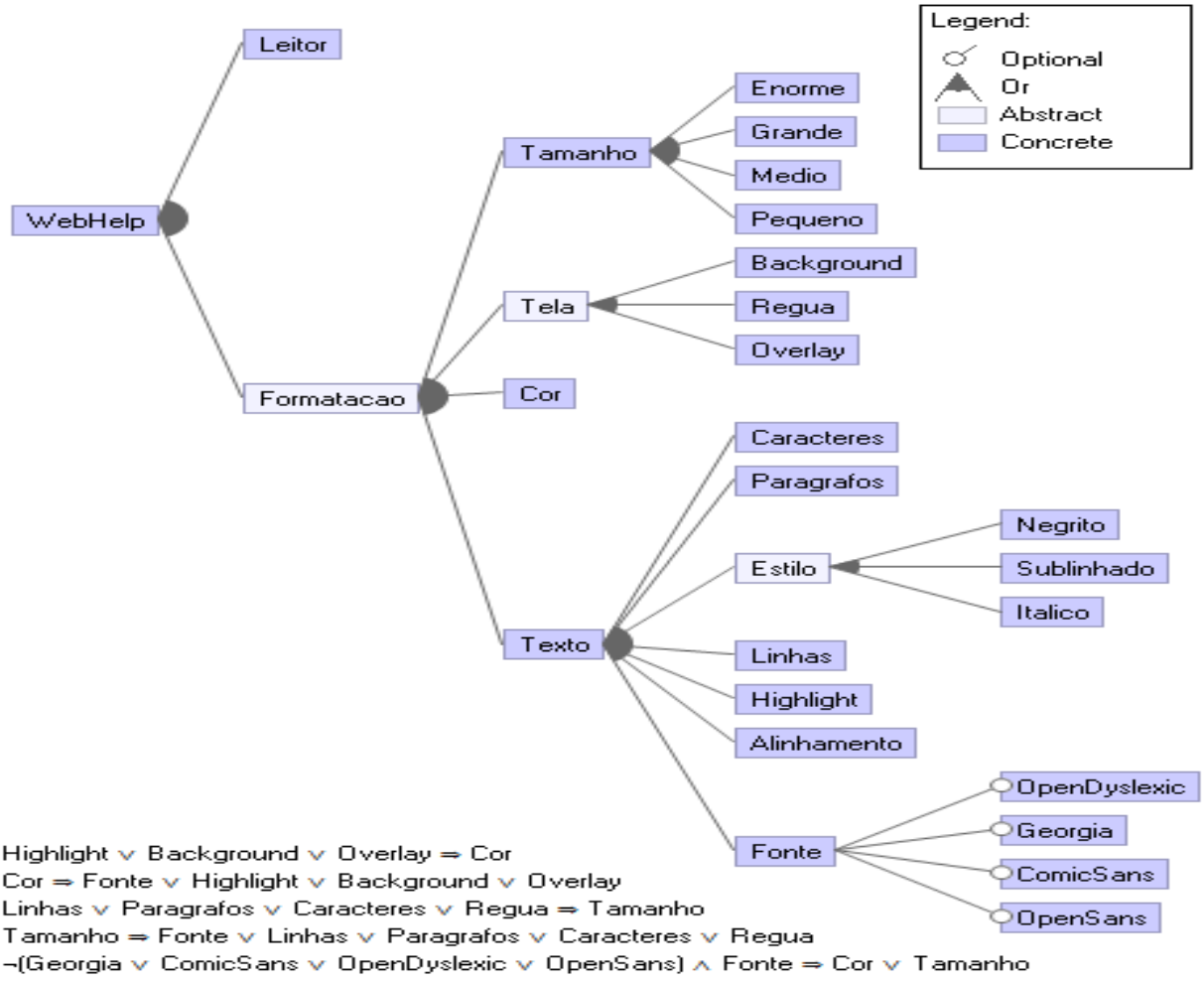
O modelo de características é apresentado na Figura 6.2, a qual possui 28 *features*, sendo 3 *features* abstratas e 25 *features* concretas. Os trechos de código responsáveis por aplicar modificações de estilo nas páginas Web encontram-se nas *features* folhas. Dessa forma, ao selecionar uma *feature* não-folha, por exemplo, a *feature* `WebHelp`, é necessário selecionar ao menos uma de suas *features* filhas `Leitor` ou `Formatação`. No caso da *feature* `Formatação` (*feature* abstrata), suas *features* filhas devem ser selecionadas até que uma *feature* folha seja selecionada, respeitando as restrições impostas no modelo de características. Por exemplo, para derivar um sistema que possua funções que permitam modificar a cor da fonte e do fundo da página Web, devem ser selecionadas as *features* `WebHelp`, `Formatação`, `Texto`, `Cor`, `Fonte` e `Background`.

Seguindo o relacionamento entre as *features* da LPS, é possível rastrear as *features* concretas na estrutura utilizada na arquitetura da LPS, em que foram utilizadas *layers* (diretórios) nas tecnologias OC e MCA e pacotes na tecnologia OA. Além disso, cada *feature* é comum ao *plug-in* `WebHelpDyslexia`[23], utilizado na atividade de implementação de domínio, onde seus artefatos de software foram extraídos para compor as *features* da `LPS WebHelp`. Uma vez que as *features* do *plug-in* `WebHelpDyslexia` foram extraídas, foi acrescentado o recurso alinhamento. Mesmo que esse recurso não seja utilizado pela maioria

[21] Disponível em: <https://featureide.github.io/>
[22] Disponível em: <http://www.eclipse.org/downloads/eclipse-packages/>
[23] Disponível em: <https://goo.gl/kdDm9A>

dos sistemas, ele é considerado importante por permitir que o texto seja alinhado à direita, à esquerda ou justificado, permitindo mais concentração no texto da página Web (AVELAR et al., 2015; eMAG, 2014).

Figura 6.2 - Modelo de Domínio (Modelo de características)



6.5 Implementação de Domínio

O *plug-in* `WebHelpDyslexia` é uma extensão do Google Chrome que visa facilitar a leitura de páginas Web por pessoas com dislexia. Essas pessoas podem ter dificuldades relacionadas à leitura e à compreensão de textos por causa da forma como são apresentados, o que inclui o tipo de fonte, o espaçamento e tamanho do texto, as palavras de difícil compreensão e a dificuldade de concentração em textos longos (AVELAR et al., 2015). Para exemplificar a sua forma de utilização, na Figura 6.3, é apresentada parte de uma página Web na sua formatação original e, na Figura 6.4, é apresentada a mesma página após selecionadas as funções Fonte do Texto, Tamanho do Texto, Cor de *Background* e Régua, disponíveis no *menu* do sistema de software `WebHelpDyslexia`. Com essa configuração, o corpo do texto

(tamanho e estilo da fonte) e a cor de fundo foram alterados, bem como o surgimento de uma régua, que destaca apenas uma parte da tela, escurecendo o restante dela.

Originalmente, o sistema de software `WebHelpDyslexia` foi desenvolvido utilizando as linguagens HTML (*HyperText Markup Language*), CSS (*Cascading Style Sheets*) e JavaScript, pois são tecnologias padrão para o desenvolvimento de extensões para o navegador Google Chrome. No entanto, neste trabalho, esse sistema foi implementado como uma aplicação *desktop* em linguagem Java para tornar-se compatível com as tecnologias de gerenciamento de variabilidades escolhidas para a obtenção de uma LPS.

Figura 6.3 - Página Web antes da Utilização do *Plug-in*

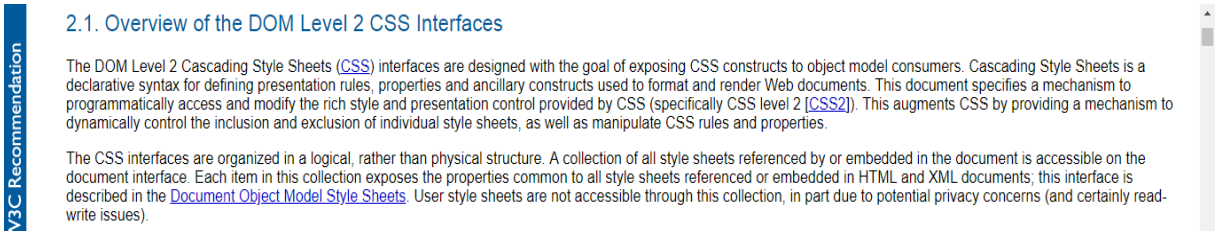
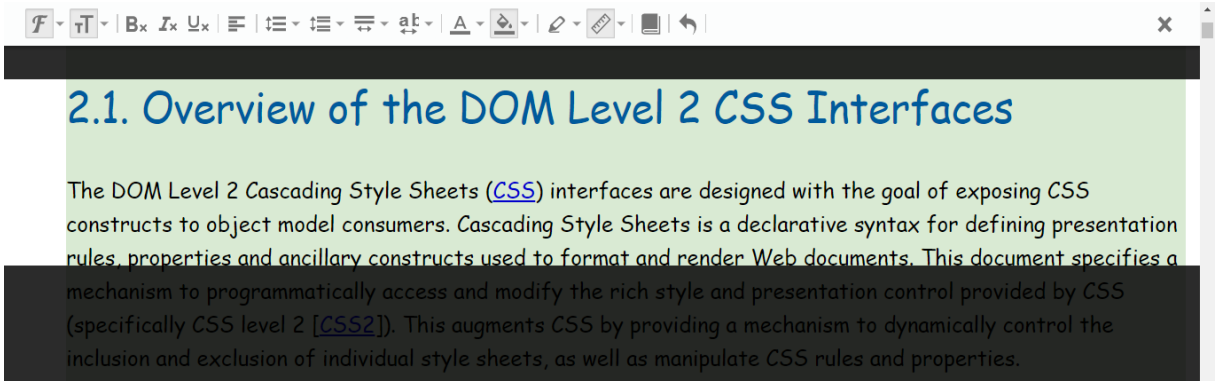


Figura 6.4 - Página Web com o *Plug-in* Ativado e com Configuração Aplicada



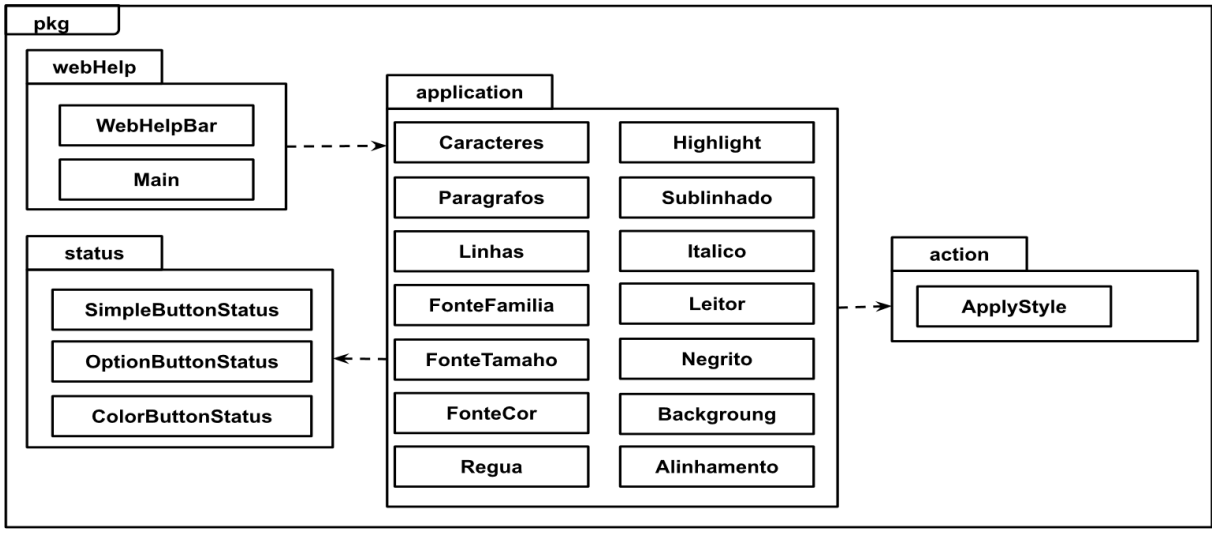
6.5.1 Implementação do *Plug-in* `WebHelpDyslexia` em Java

O desenvolvimento do *plug-in* `WebHelpDyslexia` para *desktop* consistiu na renderização de páginas Web na aplicação Java. Esse processo é justificável pois o desenvolvimento de extensões do Google Chrome utiliza, por *default*, HTML, CSS e JavaScript. Dessa forma, para implementar, foi utilizada a tecnologia JavaFX, um conjunto de gráficos e de pacotes de mídia que permite a criação e a implantação de aplicativos *rich client* que operam de forma consistente em diferentes plataformas (ORACLE, 2014). De modo a renderizar o conteúdo HTML de URLs locais ou remotas no JavaFX, há a API (*Application Programming Interface*) `WebEngine`. Com essa API, pode-se "carregar" o modelo de objeto de documento (DOM - *Document Object Model*) da página atual como conteúdo XML,

seguindo as APIs baseadas em padrões do W3C (*World Wide Web Consortium*) para Java. Assim, por meio dessa API, a página Web pode ser "percorrida" a fim de aplicar as modificações de estilo escolhidas no *menu* do sistema de software `WebHelpDyslexia`.

A estrutura do sistema de software `WebHelpDyslexia` possui 4 pacotes (Figura 6.5): i) `webHelp`; ii) `action`; iii) `application`; e iv) `status`. Nas classes do pacote `webHelp`, o sistema de software `WebHelpDyslexia` tem início. A classe `Main` é responsável por instanciar objetos das classes `WebHelpBar`, adicionando-os à `Scene`. Na classe `WebHelpBar`, são criados os botões para seleção dos recursos do sistema de software `WebHelpDyslexia`. A classe instancia objetos referentes às classes do pacote `application` com o estilo a ser aplicado à página Web. Em cada botão, há um *listener* responsável por perceber as ações relacionadas a ele. Dessa forma, quando um botão é selecionado, seu novo *status* é alterado e armazenado nas classes referentes ao pacote `status`. Finalmente, as regras de estilo são aplicadas pela classe pertencente ao pacote `action`.

Figura 6.5 - Estrutura do Sistema de Software `WebHelpDyslexia`



6.5.2 Implementação da `LPS WebHelp` em Orientação a Aspectos

Na implementação da LPS em OA[24], foi utilizada a linguagem AspectJ, em que as *features* concretas presentes no modelo de características são representadas como aspectos, sendo necessário para o *plug-in* FeatureIDE interpretar quais aspectos são referentes às *features* selecionadas para compor o produto final. A execução do código da `LPS WebHelp` é iniciada na classe `Main.java` (Código 6.1). Nessa classe, a página Web é carregada na *view* e as

[24] Código fonte da LPS desenvolvida em OA disponível em: < https://github.com/LuanaAlmeidaMartins/WebHelpAspectJ.git>

funções referentes às *features* selecionadas no modelo de características são adicionadas ao *menu* da `LPS WebHelp` (linhas 8, 9 e 12). Para tornar isso possível, os aspectos correspondentes às *features* são chamados após iniciar o *menu*. Assim, os aspectos devem possuir o mesmo nome que as *features*. No Código 6.2, é apresentado o "casamento" entre as *features* selecionadas e os aspectos, por meio do aspecto `Sublinhado.aj` (ao selecionar a *feature* `Sublinhado`, o aspecto `Sublinhado.aj` é iniciado).

Código 6.1 - Classe `Main.java`

```
 1 public class Main extends Application {
 2
 3     @Override
 4     public void start(final Stage stage) {
 5          (…)
 6
 7          // Carrega a página Web na WebEngine
 8          final WebEngine webEngine = browser.getEngine();
 9          webEngine.load(homePageUrl);
10
11          // Cria o menu
12          WebHelpBar webHelpBar = new WebHelpBar(browser);
13          (…)
14     }
15
16     public static void main(String[] args){
17          launch(args);
18     }
19 }
```

Código 6.2 - Aspecto `Sublinhado.aj`

```
 1 public aspect Sublinhado {
 2
 3     final String ID = "Sublinhado";
 4
 5     // Carrega funcao no menu
 6     after(): initialization(WebHelpBar.new(WebView)) {
 7          SimpleButton a = new SimpleButton (featureName);
 8          a.actionButton();
 9      }
10
11     // Executa acao quando clicado
12     after(SimpleButton button) : target(button)
           && call(private void applyStyle(..)){
13          if(button.getSimpleButton().getButtonID().equals(ID)) {
14            WebHelpBar.applyButtonStatus.setFontStyle(
                  button.getSimpleButton().getStyle(),
                  handle.getSimpleButton().isActive());
15          }
16     }
17 }
```

Além disso, no Código 6.2, pode-se perceber que o aspecto possui dois adendos, o primeiro "carrega" o botão referente à *feature* no *menu* da `LPS WebHelp` (linhas 6 - 9) e o segundo executa a ação da *feature* quando o botão é selecionado (linhas 12 - 16). No geral, o "carregamento" dos botões no *menu* difere-se pelo tipo de botão utilizado. Para as *features* `Alinhamento`, `Itálico`, `Negrito`, `Sublinhado` e `Leitor`, são utilizados botões simples do tipo `Button`. Para as *features* que possuem restrições transversais com a *feature* `Cor` (*features* `Fonte`, `Background`, `Highlight` e `Overlay`), são utilizados botões do tipo `ColorPicker`, responsável por disponibilizar a paleta de cores. Para as *features* que possuem restrições transversais com a *feature* `Tamanho` (*features* `Caracteres`, `Linhas`, `Parágrafos` e `Fonte`), são utilizados botões do tipo `RadioButton`, em que apenas uma opção pode ser selecionada por vez.

Desse modo, para as *features* que não possuem restrições transversais, são instanciados objetos da classe `SimpleButton.java`, como apresentado no Código 6.2 (linha 7). Diferentemente, a implementação das *features* com restrições transversais é apresentada no Código 6.3 (*e.g.*, aspecto `Linhas.aj`). Para as *features* com restrições transversais com as *features* `Tamanho` e `Cor`, são instanciados objetos das classes `SizeButton.java` ou `ColorButton.java`, respectivamente (linha 4). Dessa forma, pode-se perceber que o aspecto possui três adendos, o primeiro "carrega" o botão referente à *feature* no *menu* da `LPS WebHelp` (linha 7), o segundo "carrega" as opções referentes ao botão (linhas 13 - 16), identificadas por meio do ponto de corte (linha 10), e o terceiro executa a ação correspondente a opção selecionada no botão (linhas 19 - 21).

No Código 6.4, é apresentada a identificação das funções da `LPS WebHelp` e a aplicação do estilo correspondente na página Web, por meio da classe `SimpleButton.java`. As funções são identificadas por meio de um *listener* que captura a ação do clique nos botões (linhas 6 - 10). Inicialmente, cada botão possui o *status* desativado e, ao ser clicado, o *status* passa a ser ativo e o seu estilo é aplicado (linhas 12 -16). Ao chamar o método `applyStyle` (linha 15), o adendo `call(private void applyStyle(..))`, definido em cada aspecto (Código 6.2 - linha 12 e Código 6.3 - linha 19), é executado para aplicar o estilo específico de cada botão. Para essa aplicação, é encontrada a *string* correspondente ao nome do botão em um documento *properties* e a árvore DOM da aplicação é percorrida para acrescentar essa *string* a cada um de seus elementos.

Código 6.3 - Aspecto `Linhas.aj`

```
1  public aspect Linhas {
2
3      final String ID = "Linhas";
4      SizeButton opcaoTamanho = new SizeButton(ID);
5
6      // Carrega RadioButton no menu
7      after(): initialization(WebHelpBar.new(WebView)) {}
8
9      // Cria ponto de corte referente as opções de Tamanho
10     pointcut Tamanho(): within(Pequeno) || within(Medio) ||
                           within(Grande) || within(Enorme);
11
12     // Carrega as opções de tamanho no RadioButton
13     after(): Tamanho(){
14         opcaoTamanho.opcao(thisJoinPointStaticPart);
15         opcaoTamanho.actionButton();
16     }
17
18     // Executa acao quando clicado
19     after(SizeButton button) : target(button)
           && call(private void applyStyle(..)){
20         (…)
21     }
22 }
```

Código 6.4 - Classe `SimpleButton.java`

```
1  public class SimpleButton {
2      (…)
3
4      // Identifica clique e aplica estilo do botão
5      public void action() {
6          WebHelpBar.webEngine.getLoadWorker().stateProperty()
                   .addListener((obs, oldValue, newValue) -> {
7              if (newValue == State.SUCCEEDED) {
8                  Document doc = WebHelpBar.webEngine.getDocument();
9                  WebHelpBar.applyButtonStatus =
                       new ApplyButtonStatus(doc);
10                 botao.setOnMouseClicked(new
                       EventHandler<MouseEvent>() {
11
12                     @Override
13                     public void handle(MouseEvent event) {
14                         status.setStatus();
15                         applyStyle();
16                     }
17                 });
18             }
19         });
20         WebHelpBar.hbox.getChildren().add(botao);
21     }
22
23     private void applyStyle() {}
24 }
```

6.5.3 Implementação da `LPS WebHelp` em Orientação a Características

Para a implementação da LPS em OC [25], foi utilizada a linguagem Jakarta, em que existe um diretório (*layer*) com refinamentos e classes para cada *feature* concreta. Dessa forma, o *plug-in* FeatureIDE consegue interpretar quais arquivos são referentes às *features* selecionadas no modelo de características. A execução do código da `LPS WebHelp` é iniciada no diretório `WebHelp` que contém a classe `Main.jak` (Código 6.5), responsável por "carregar" a página Web na *view* e o *menu* com as funções referentes às *features* selecionadas (linhas 8, 9 e 12). Para isso, foi criado o método `createWebHelpBar()` na classe `Main.jak` (linha 17), sendo chamado após iniciar o *menu* da `LPS WebHelp` para ser refinado pelas *features* (linha 13). O refinamento desse método é exemplificado no Código 6.6 por meio da *feature* `Sublinhado`. Cabe ressaltar que, os refinamentos mantêm o nome da classe de origem, ou seja, o diretório `Sublinhado` contém o refinamento `Main.jak`.

Código 6.5 - Classe `Main.jak`

```
1  public class Main extends Application {
2
3      @Override
4      public void start(final Stage stage) {
5           (…)
6
7           // Carrega a página Web na WebEngine
8           final WebEngine webEngine = browser.getEngine();
9           webEngine.load(homePageUrl);
10
11          // Cria o menu
12          WebHelpBar webHelpBar = new WebHelpBar(browser);
13          new Main().createWebHelpBar();
14          (…)
15     }
16
17     public void createWebHelpBar() {}
18
19     public static void main(String[] args){
20          launch(args);
21     }
22 }
```

Além disso, no Código 6.6, é possível perceber a instanciação de um objeto da classe `SimpleButton.jak` (linha 7). Da mesma forma como implementado em OA, o "carregamento" dos botões no *menu* difere-se pelo tipo de botão utilizado influenciado pelo tipo de restrição transversal imposta no modelo de características. Para as *features* sem restrição

[25] Código fonte da LPS desenvolvida em OC disponível em: < https://github.com/LuanaAlmeidaMartins/WebHelpAHEAD.git>

transversal, com restrição transversal à *feature* `Cor` e com restrição transversal à *feature* `Tamanho`, é utilizado um botão `Button`, um botão `ColorPicker` e um botão `RadioButton`, respectivamente.

Código 6.6 - Refinamento da Classe `Main.jak` no Diretório `Sublinhado`

```
1  public refines class Main {
2
3      final String ID = "Sublinhado";
4
5      public void createWebHelpBar() {
6          Super().createWebHelpBar() ;
7          SimpleButton a = new SimpleButton (ID);
8          a.action();
9      }
10 }
```

Dessa forma, para as *features* sem restrição, são instanciados objetos da classe `SimpleButton.jak`. Como as *features* sem restrição tranversal são filhas da *feature* `Texto`, essa classe foi definida no diretório `Texto` (Código 6.7). Similarmente, no diretório `Tamanho`, foi definida a classe `SizeButton.jak` e, no diretório `Cor`, foi definida a classe `ColorButton.jak`. Desse modo, o objeto da classe `SimpleButton.jak`, instanciado no refinamento da classe `Main.jak` (Código 6.6 - linha 7), realiza a chamada do método `action()`. Nesse método, definido no Código 6.7, são identificadas as chamadas de funções por meio de um *listener*, capturando a ação do clique nos botões (linhas 6 - 10). Quando clicado, o *status* do botão é configurado para ativo e é feita a instanciação da classe `ApplySimpleButton.jak` para aplicar o estilo (linhas 12 -16).

Assim, para aplicar estilos referentes às *features* sem restrição transversal, a classe `ApplySimpleButton.jak` é criada no diretório `Texto`, refinada nos diretórios `Alinhamento`, `Negrito`, `Sublinhado`, `Itálico` e `Leitor`. Para as *features* com restrição transversal com a *feature* `Cor`, a classe `ApplyColorButton.jak` é criada no diretório `Cor`, refinada nos diretórios `Fonte`, `Background`, `Highlight` e `Overlay`. Para as *features* com restrição transversal com a *feature* `Tamanho`, a classe `ApplyTamanhoButton.jak` é criada no diretório `Tamanho`, refinada nos diretórios `Fonte`, `Regua`, `Paragrafos`, `Linhas` e `Caracteres`. No Código 6.8, a classe `ApplySimpleButton.jak` no diretório `Texto` é apresentada. Nessa classe, têm-se a criação de um botão simples (linhas 3, 5 - 7) e do método `applyStyle()` a ser refinado pelas *features* sem restrições transversais (linha 9). No Código 6.9, o método `applyStyle()` é refinado no diretório `Sublinhado` para a aplicação de seu estilo (linhas 3 - 8).

Código 6.7 - Classe `SimpleButton.jak` no Diretório `Texto`

```
1  public class SimpleButton {
2      (…)
3
4      // Identifica clique e aplica estilo do botão
5      public void action() {
6          WebHelpBar.webEngine.getLoadWorker().stateProperty()
                       .addListener((obs, oldValue, newValue) -> {
7              if (newValue == State.SUCCEEDED) {
8                  Document doc =
   WebHelpBar.webEngine.getDocument();
9                  WebHelpBar.applyButtonStatus =
                       new ApplyButtonStatus(doc);
10                 botao.setOnMouseClicked(new
                       EventHandler<MouseEvent>() {
11
12                     @Override
13                     public void handle(MouseEvent event) {
14                         status.setStatus();
15                         new ApplySimpleButton(status).simple();
16                     }
17                 });
18             }
19         });
20     }
21 }
```

Código 6.8 - Classe `ApplySimpleButton.jak` no Diretório `Texto`

```
1  public class ApplySimpleButton {
2
3      SimpleButtonStatus simpleStatus;
4
5      public ApplySimpleButton(SimpleButtonStatus status) {
6          this.simpleStatus = status;
7      }
8
9      public void applyStyle() { (…)}
10 }
```

Código 6.9 - Classe `ApplySimpleButton.jak` no Diretório `Sublinhado`

```
1  public refines class ApplySimpleButton {
2
3      public void applyStyle() {
4          if(simpleStatus.getButtonID().equals(ID)) {
5              WebHelpBar.applyButtonStatus.setFontStyle(
                   simpleStatus.getStyle(),simpleStatus.getStatus());
6          }
7          Super().simple();
8      }
9  }
```

6.5.4 Implementação da `LPS WebHelp` em Módulos de Características Aspectuais

Inicialmente, a `LPS WebHelp` em MCA [26] foi estruturada utilizando diretórios como na tecnologia OC. Além disso, as classes e os seus refinamentos foram mantidos e, posteriormente, os aspectos foram inseridos nessa estrutura. Uma vez que a tecnologia OC lida com extensões heterogêneas, ou seja, extensões que adicionam diferentes partes de código em diferentes pontos de extensão, foram analisadas quais extensões de classes seriam mantidas. Como resultado, foi identificado que as extensões da classe `Main.jak` do diretório `WebHelp` deveriam ser mantidas nos diretórios `Texto`, `Tamanho` e `Cor`. A heterogeneidade dessas extensões está relacionada ao tipo de botão adicionado ao *menu* da `LPS WebHelp` (`Button`, `RadioButton` e `ColorPicker`, respectivamente). Além disso, as classes `SimpleButton.jak`, `SizeButton.jak` e `ColorButton.jak` foram mantidas nos seus diretórios. Visto que a *feature* `Tela` é abstrata, suas *features* filhas também possuem refinamentos da classe `Main.jak`, sendo mantidos.

Posteriormente, foram analisados os refinamentos da classe `Main.jak` nos diretórios `Texto`, `Tamanho` e `Cor`. Os atributos que recebem objetos da classe `SizeButton.jak` (*e.g.*, `paragrafos`, `caracteres` e `linhas`) e da classe `ColorButton.jak` (*e.g.*, `highlight` e `background`), utilizados nas *features* com restrições transversais com as *features* `Tamanho` e `Cor`, são definidos no refinamento da classe `Main.jak`. Dessa forma, é necessário identificar quais atributos foram iniciados na `LPS WebHelp` para os botões e suas opções serem adicionados ao *menu*. Para isso, foram utilizadas extensões heterogêneas em suas *features* filhas, sendo as *features* `Pequeno`, `Medio`, `Grande` e `Enorme` filhas da *feature* `Tamanho`. Similarmente, a *feature* `Fonte` possui as *features* filhas `Georgia`, `ComicSans`, `OpenSans`, e `OpenDyslexic` e possui restrição transversal com as *features* `Tamanho` e `Cor`. Portanto, suas *features* filhas também foram mantidas como implementadas em OC, para identificar o valor inicial de seus atributos.

O refinamento da classe `Main.jak` no diretório `Texto` não apresenta declarações de atributos. Uma vez que são utilizados botões simples, não há opções para serem adicionadas ao botão. Dessa forma, o código é replicado nas *features* `Sublinhado`, `Italico`, `Negrito`, `Paragrafos`, `Caracteres`, `Linhas`, `Alinhamento` e `Highlight` para sua inserção no *menu* da `LSP WebHelp`. Isto é, essas *features* fazem o uso de extensões homogêneas, pois

[26] Código fonte da LPS desenvolvida em MCA disponível em: < https://github.com/LuanaAlmeidaMartins/WebHelpAFM.git >

são extensões que adicionam o mesmo trecho de código em diferentes pontos de extensão. Desse modo, os refinamentos da classe `Main.jak` foram substituídos por aspectos. Da mesma forma, a *feature* `Leitor` foi convertida para aspecto, visto que não possui declarações de atributos referentes às restrições transversais. Dessa forma, os aspectos tornaram-se responsáveis por interceptar o método `action()` definido na classe `SimpleButton.jak`, tornando desnecessários a classe `ApplySimpleButton.jak` e seus refinamentos.

Nesse sentido, as classes `ApplySizeButton.jak` e `ApplyColorButton.jak` também são refinadas de modo a permitir a reescrita do método `action()` definido nas classes `SizeButton.jak` e `ColorButton.jak`. Logo, essas classes e seus refinamentos foram removidos e aspectos foram adicionados para reescrever o método `action()`. Assim, foram inseridos aspectos nos diretórios `Fonte`, `Overlay`, `Regua` e `Background`, os quais possuem implementações de refinamentos e de aspectos.

Portanto, a estrutura da LPS em MCA é organizada da seguinte forma. As *features* `Overlay`, `Background`, `Regua`, `Caracteres`, `Paragrafos`, `Fonte`, `Negrito`, `Sublinhado`, `Itálico`, `Alinhamento`, `Linhas`, `Leitor` e `Highlight` são módulos de características aspectuais, pois essas *features* foram modularizadas em módulos de características e contém implementações de aspectos. As *features* `OpenSans`, `ComicSans`, `Georgia`, `OpenDyslexic`, `Pequeno`, `Medio`, `Grande`, `Enorme`, `WebHelp`, `Tamanho`, `Cor` e `Texto` são módulos de características, uma vez que não foram inseridos aspectos nesses módulos.

Para exemplificar como a `LPS WebHelp` em MCA foi implementada, foi selecionada a *feature* `Paragrafos` para a derivação de um produto, requerendo a seleção das *features* `WebHelp`, `Texto`, `Tamanho`, e opcionalmente a seleção de ao menos uma das *features* `Pequeno`, `Medio`, `Grande` e `Enorme`. No Código 6.5, foi apresentada a implementação da classe `Main.jak` no diretório `WebHelp`; essa implementação permaneceu a mesma. No Código 6.10, a classe `Main.jak` é refinada no diretório `Tamanho`, em que os atributos referentes às *features* com restrições transversais com a *feature* `Tamanho` são declarados (linha 2) e adicionados na classe `Main.jak`. Posteriormente, no aspecto `Paragrafos.aj`, implementado no Código 6.11, é atribuído valor inicial ao atributo `paragrafos` (linhas 3 - 5). Nesse aspecto, é utilizada a classe `Main$$Tamanho.jak` (linha 3) referente ao refinamento da classe `Main.jak` no diretório `Tamanho`. No Código 6.12, as *features* filhas da *feature* `Tamanho` refinam a classe `Main.jak`, cujo atributo `paragrafos` possui valor

inicial para o qual são adicionadas as opções Pequeno, Médio, Grande e Enorme. Nesse momento, é realizada a verificação de quais atributos recebem objetos da classe `SizeButton.jak` (Código 6.10 - linha 2) são utilizados para as suas opções serem adicionadas ao botão (Código 6.12 - linhas 4 - 8). Além disso, cada opção chama o método `action()`, interceptado pelo aspecto `Paragrafos.aj`, para executar a sua ação.

Código 6.10 - Refinamento da Classe `Main.jak` no Diretório `Tamanho`

```
1  public refines class Main {
2      SizeButton paragrafos = null, linhas = null,
           caracteres = null, fonte = null;
3
4      public void createWebHelpBar() {
5          Super().createWebHelpBar() ;
6      }
7  }
```

Código 6.11 - Aspecto `Paragrafos.aj`

```
1  public aspect Paragrafos {
2
3      after(): execution(void Main$$Tamanho.createWebHelpBar()){
4          paragrafos = new SizeButton ("Paragrafos");
5      }
6
7      after(SizeButton button): target(button) &&
           call(private void action(..)) {
8          (…)
9      }
10 }
```

Código 6.12 - Refinamento da Classe `Main.jak` no Diretório `Grande`

```
1  public refines class Main {
2     Super().createWebHelpBar() ;
3
4      if(paragrafos != null) {
5          paragrafos.opcao("Grande");
6          paragrafos.action() ;
7      }
8      (…)
9  }
```

6.6 Exemplo de Derivação de Produto

Quanto à atividade de Teste de Domínio, foram realizados testes caixa preta apenas para verificar o funcionamento dos recursos. Na Figura 6.6, é apresentado um produto derivado da `LPS WebHelp` com todos os seus recursos (barra de ícones). Na Figura 6.7, é apresentado

outro produto derivado da LPS WebHelp com recursos relacionados à fonte (cor, tamanho e família) e à cor de fundo (barra de ícones).

Figura 6.6 - Produto Derivado da LPS com Todos os Recursos

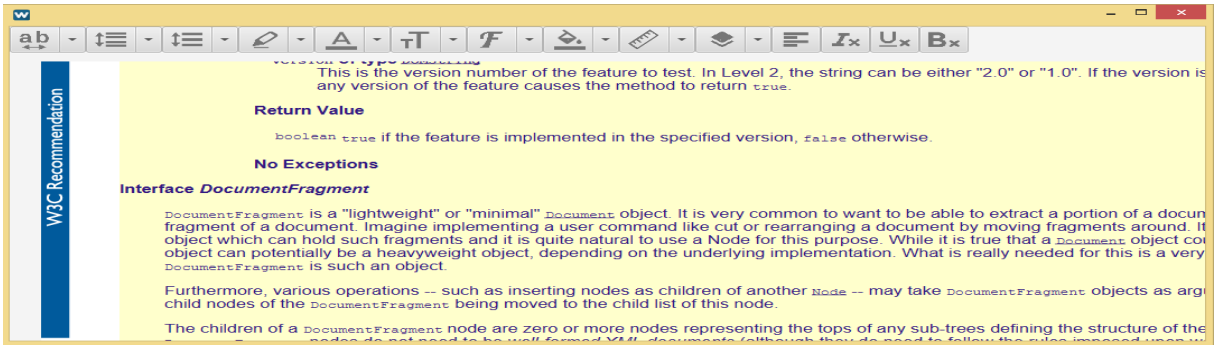
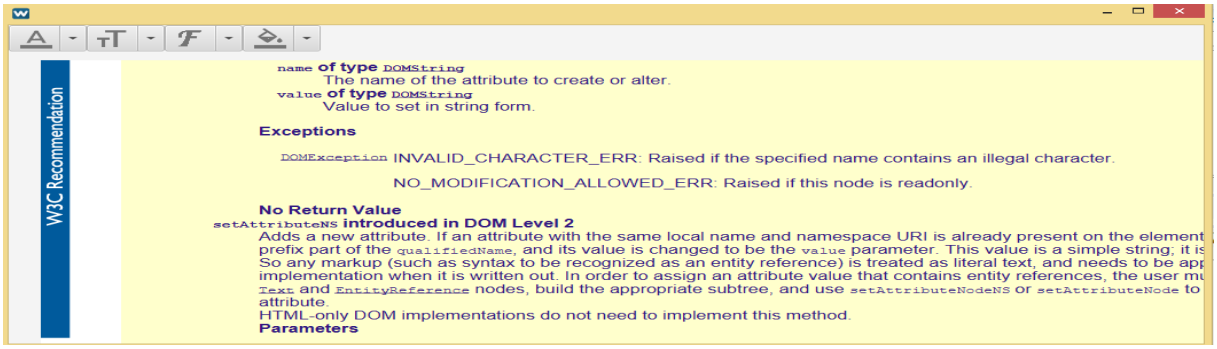


Figura 6.7 - Produto Derivado da LPS com os Recursos para a Combinação entre Fonte e Fundo da Página



6.7 Análise Estatística

De modo a investigar se existe diferença significativa entre a modularidade da LPS implementada com diferentes tecnologias de gerenciamento de variabilidades, foram consideradas medidas relacionadas às propriedades de acoplamento e de coesão de sistemas de software. Desse modo, foram estabelecidas as hipóteses de pesquisa

> **H_0**: A modularidade da LPS WebHelp desenvolvida com as tecnologias de Orientação a Características, de Orientação a Aspectos e de Módulos de Características Aspectuais são equivalentes.

> **H_1**: A modularidade da LPS WebHelp desenvolvida com as tecnologias de Orientação a Características, de Orientação a Aspectos e de Módulos de Características Aspectuais não são equivalentes.

As propriedades de acoplamento e de coesão são frequentemente utilizadas para mensurar a modularidade em sistemas de software (BAVOTA et al., 2010; BAVOTA et al., 2012; BECK et al., 2011; COLANZI, 2012). No entanto, vale ressaltar que outras propriedades

podem ser utilizadas para mensurar a modularidade, por exemplo, o entrelaçamento e o espalhamento de código de *features*. Além disso, há diversas medidas de software que podem ser utilizadas para mensurar o acoplamento e a coesão, porém apenas seis medidas foram escolhidas para a realização deste trabalho. O acoplamento entre as *features* da LPS é dado pela Equação 6.1, em que é feita a média aritmética dos valores obtidos para as medidas DepIn, DepOut e SFC. A coesão das *features* da LPS é dada pela Equação 6.2, em que é feita a média aritmética das medidas EFD, IFD e LCC. Como a medida LCC mensura a falta de coesão, ela possui crescimento contrário as medidas IFD e EFD; por isso, essa medida é subtraída na equação. Além disso, de modo a caracterizar o tamanho da LPS, foram coletadas as medidas LOC, NOA e NOO.

$$Acoplamento = \frac{DepIn + DepOut + SFC}{3} \tag{6.1}$$

$$Coesão = \frac{EFD + IFD - LCC}{3} \tag{6.2}$$

Após a coleta dos dados, a sua distribuição foi verificada por meio do teste de normalidade de Kolmogorov-Smirnov (LILLIEFORS, 1967), para cada medida de software e para os agrupamentos das medidas nas propriedades de acoplamento e de coesão. Posteriormente, para investigar se existe diferença significativa entre a modularidade da LPS desenvolvida com as três tecnologias de gerenciamento de variabilidades, foi aplicado o teste de hipóteses de Friedman (FRIEDMAN, 1940) e realizado o *post hoc* de Conover (CONOVER, 1999) para identificar entre quais pares de tecnologias foi constatada a diferença significativa. Para investigar se existe diferença significativa entre as tecnologias de gerenciamento de variabilidades, são considerados o valor das medidas de software de forma individual e o seu agrupamento nas propriedades de acoplamento e de coesão.

Esta seção está organizada da seguinte forma. Na Seção 6.7.1, são apresentadas as análises descritivas, os testes de normalidade e de hipóteses para cada uma das medidas de software. Na Seção 6.7.2, são apresentadas as análises descritivas, os testes de normalidade e de hipóteses para o agrupamento as medidas de software nas propriedades de acoplamento e de coesão.

6.7.1 Análise das Medidas de Software

O conjunto de dados do estudo é constituído pela `LPS WebHelp` desenvolvida em três tecnologias de gerenciamento de variabilidades. Desse modo, o valor das medidas de software

é coletado em três sistemas de software que implementam a mesma LPS. Dado que a unidade de decomposição compartilhada pelas tecnologias de gerenciamento de variabilidades é a *feature*, as medidas são coletadas e analisadas em nível de *feature*. Em outras palavras, o sistema desenvolvido em OA possui a mesma modularidade em nível de *features* que o sistema desenvolvido em OC e em MCA. No entanto, cada *feature* possui diferentes componentes (classes, interfaces, aspectos e refinamentos). Além disso, cabe ressaltar que o valor das medidas de software é calculado para *features* concretas, em que a `LPS WebHelp` totaliza 25 *features* concretas. Os dados brutos da coleta do valor das medidas de software são apresentados no APÊNDICE C.

Para descrever e caracterizar a amostra, foram utilizadas medidas de tendência central e medidas de dispersão. As medidas de tendência central incluem a média e mediana e quartis. As medidas de dispersão incluem o desvio padrão, a variância, o valor máximo e o valor mínimo. O valor médio das medidas utilizadas é apresentado na Tabela 6.2, em que o maior valor entre as tecnologias está grafado em negrito e o menor valor entre as tecnologias é grafado em sublinhado.

Tabela 6.2 - Valor Médio das Medidas de Software para cada Tecnologia

Tecnologias	LOC	NOA	NOO	EFD	IFD	LCC	DepIn	DepOut	SFC
OC	**34,84**	1,4	**3,92**	0,14	0,04	2,36	1,36	1,36	**0,08**
OA	21,36	**2,00**	3,72	0,12	0,05	**3,48**	**2,48**	**2,48**	0,06
MCA	28,40	1,80	3,80	**0,15**	**0,07**	2,00	1,00	1,00	0,07

Ao interpretar o valor médio obtido para as medidas de tamanho, tem-se que a LPS implementada em OC é maior em quantidade de linhas (LOC) e de operações (NOO) e é menor em quantidade de atributos (NOA). Por outro lado, a LPS implementada em OA é menor em quantidade de linhas e de operações e é a maior em quantidade de atributos. Em relação as medidas de coesão, tem-se que os valores mais próximos de 0 (zero) indicam pior coesão para EFD e IFD e melhor coesão para LCC. Desse modo, os valores maiores para IFD e EFD e menor valor para LCC foi obtido com a tecnologia MCA, indicando mais coesão. Em relação as medidas de acoplamento, tem-se que os valores mais próximos de 0 (zero) indicam menos acoplamento. Dessa forma, para as medidas DepIn e DepOut, tem-se que a LPS implementada com a tecnologia MCA é a menos acoplada e a tecnologia OA é a mais acoplada.

Adicionalmente, é possível perceber que as medidas EFD e IFD apresentaram baixa coesão para as três tecnologias de gerenciamento de variabilidades. Na medida EFD, calcula-se a razão entre a quantidade de dependências internas pela quantidade total de dependências da *feature*. Na medida IFD, calcula-se a razão entre a quantidade de dependências internas pelo total de possíveis dependências internas da *feature*. Dessa forma, como a `LPS WebHelp`

possui diversas *features* com apenas um componente, elas possuem poucas dependências internas, o que resulta em baixa coesão ao utilizar essas medidas.

Além disso, cada tecnologia de gerenciamento de variabilidades pode ser descrita em termos do valor mínimo, do 1º quartil (25%), da mediana, do 3º quartil (75%), do valor máximo e do desvio padrão (estatística descritiva). Na Tabela 6.3, na Tabela 6.4 e na Tabela 6.5, é apresentada a estatística descritiva para as tecnologias OC, OA e MCA, respectivamente.

Tabela 6.3 - Estatística Descritiva para a Tecnologia Orientação a Características

	LOC	NOA	NOO	EFD	IFD	LCC	DepIn	DepOut	SFC
Mínimo	10,00	0,00	1,00	0,00	0,00	1,00	0,00	0,00	0,02
25%	20,00	0,00	1,00	0,00	0,00	2,00	0,00	1,00	0,02
Mediana	24,00	0,00	3,00	0,00	0,00	2,00	0,00	1,00	0,06
75%	33,00	0,00	3,00	0,00	0,00	3,00	0,00	2,00	0,11
Máximo	117,00	13,00	14,00	1,00	0,25	4,00	13,00	3,00	0,23
Média	34,84	1,40	3,92	0,14	0,04	2,36	1,36	1,36	0,08
Desvio Padrão	30,31	3,45	3,75	0,31	0,08	0,08	3,58	0,64	0,06

Tabela 6.4 - Estatística Descritiva para a Tecnologia Orientação a Aspectos

	LOC	NOA	NOO	EFD	IFD	LCC	DepIn	DepOut	SFC
Mínimo	4,00	0,00	1,00	0,00	0,00	1,00	0,00	0,00	0,00
25%	6,00	0,00	1,00	0,00	0,00	2,00	0,00	1,00	0,00
Mediana	8,00	1,00	3,00	0,00	0,00	3,00	0,00	2,00	0,04
75%	8,00	2,00	3,00	0,00	0,00	3,00	5,00	2,00	0,10
Máximo	107,00	11,00	14,00	1,00	0,25	12,00	24,00	11,00	0,18
Média	21,36	2,00	3,72	0,12	0,05	3,48	2,48	2,48	0,06
Desvio Padrão	32,62	2,55	3,82	0,27	0,10	0,10	4,97	2,54	0,05

Tabela 6.5 - Estatística Descritiva para a Tecnologia Módulos de Características Aspectuais

	LOC	NOA	NOO	EFD	IFD	LCC	DepIn	DepOut	SFC
Mínimo	4,00	0,00	1,00	0,00	0,00	1,00	0,00	0,00	0,02
25%	12,00	0,00	1,00	0,00	0,00	2,00	0,00	1,00	0,02
Mediana	14,00	1,00	3,00	0,00	0,00	2,00	0,00	1,00	0,06
75%	26,00	1,00	3,00	0,00	0,00	2,00	0,00	1,00	0,072
Máximo	117,00	13,00	14,00	1,00	0,50	3,00	13,00	2,00	0,22
Média	28,40	1,80	3,80	0,15	0,07	2,00	1,00	1,00	0,07
Desvio Padrão	31,70	3,07	3,87	0,32	0,14	0,14	2,84	0,29	0,06

Por exemplo, em relação à quantidade de linhas, as *features* da tecnologia OC (Tabela 6.3) apresentam o valor mínimo de dez linhas e máximo de 117 linhas. O 1º quartil, que representa ¼ do conjunto de dados, está entre 10 linha e 20 linhas. O 3º quartil, que representa ¾ do conjunto de dados, está entre 33 linhas e 117 linhas. Esse sistema possui em média 34,84 linhas, mediana de 24 linhas e desvio padrão de 30,3, indicando alta dispersão dos valores.

Além disso, para visualizar a distribuição dos dados, na Figura 6.8, na Figura 6.9 e na Figura 6.10, são apresentados histogramas juntamente com a curva de distribuição normal esperada para cada medida para as tecnologias OC, OA e MCA, respectivamente. Os histogramas podem revelar características distintamente não-normais de uma distribuição, por exemplo, *outliers*, assimetria acentuada, lacunas ou aglomerados. Nesse sentido, é perceptível

que apenas a medida SFC não apresenta lacunas em sua distribuição, seja qual for a tecnologia mensurada (Figura 6.8(i), Figura 6.9(i) e Figura 6.10(i)). Dessa forma, é esperado que, para o restante das medidas de software coletadas, com exceção da SFC, a distribuição dos dados seja não-normal. Contudo, para confirmar a (não) normalidade da distribuição dos dados, cabe a realização dos testes de normalidade. Para verificar a distribuição dos dados, foi aplicado o teste de Kolmogorov-Smirnov (LILLIEFORS, 1967) para avaliar as hipóteses

H_0: Os dados seguem a distribuição especificada.

H_1: Os dados não seguem a distribuição especificada.

A partir desse teste, têm-se a diferença entre a função de distribuição cumulativa assumida para os dados (normal) e a função de distribuição empírica dos dados (Equação 6.3).

$$D_n = max\,|F(x) - S_n(x)| \tag{6.3}$$

sendo `F(x)` a função de distribuição cumulativa assumida para a amostra e $S_n(x)$ a função de distribuição cumulativa da amostra. Desse modo, a equação calcula a distância máxima entre a distribuição de probabilidades acumuladas teórica e observadas (MILLE, 1956).

A interpretação do valor D_n é feita por meio da tabela de valores críticos de Mille, que recebe como parâmetros o nível de confiança (`alpha = 5%`) e o tamanho da amostra (`n = 25` *features*). Desse modo, para rejeitar a hipótese nula com `(1 - alpha) * 100` de confiança, o valor D_n deve ser maior que o valor crítico que é 0,23768 (MILLE, 1956). Outra forma de interpretar o resultado do teste é por meio do *p-value*, que indica a chance do valor da estatística Komogorov-Smirnov ser igual ou maior do que o observado. Dessa forma, se o *p-value* for menor que o nível de confiança, então as amostras vêm de populações com distribuições diferentes da normal.

O teste de Kolmogorov-Smirnov foi aplicado apenas nas medidas de acoplamento e de coesão, visto que essas são as medidas utilizadas para mensurar propriedades relacionadas à modularidade da LPS. Os resultados obtidos são apresentados na Tabela 6.6, em que as medidas que possuem uma distribuição normal são grafadas em negrito. Isto é, apenas a medida SFC apresenta uma distribuição normal em todas as tecnologias.

Tabela 6.6 - Valores de D_n para o Teste Kolmogorov-Smirnov

Tecnologias	EFD	IFD	LCC	DepIn	DepOut	SFC
OC	0,47829 ($2{,}155 \times 10^{5}$)	0,48191 ($1{,}812 \times 10^{5}$)	0,3538 (0,003827)	0,48795 ($1{,}352 \times 10^{5}$)	0,3538 (0,003817)	**0,19625 (0,2907)**
OA	0,47407 ($2{,}635 \times 10^{5}$)	0,48749 ($1{,}383 \times 10^{5}$)	0,37509 (0,001762)	0,30881 (0,01699)	0,37509 (0,001762)	**0,17334 (0,4403)**
MCA	0,4828 ($1{,}735 \times 10^{5}$)	0,47995 ($1{,}99 \times 10^{5}$)	0,46 ($5{,}084 \times 10^{5}$)	0,47748 ($2{,}241 \times 10^{5}$)	0,46 ($5{,}084 \times 10^{5}$)	**0,23357 (0,1307)**

Figura 6.8 - Distribuição dos Dados Amostrais para a Tecnologia Orientação a Características

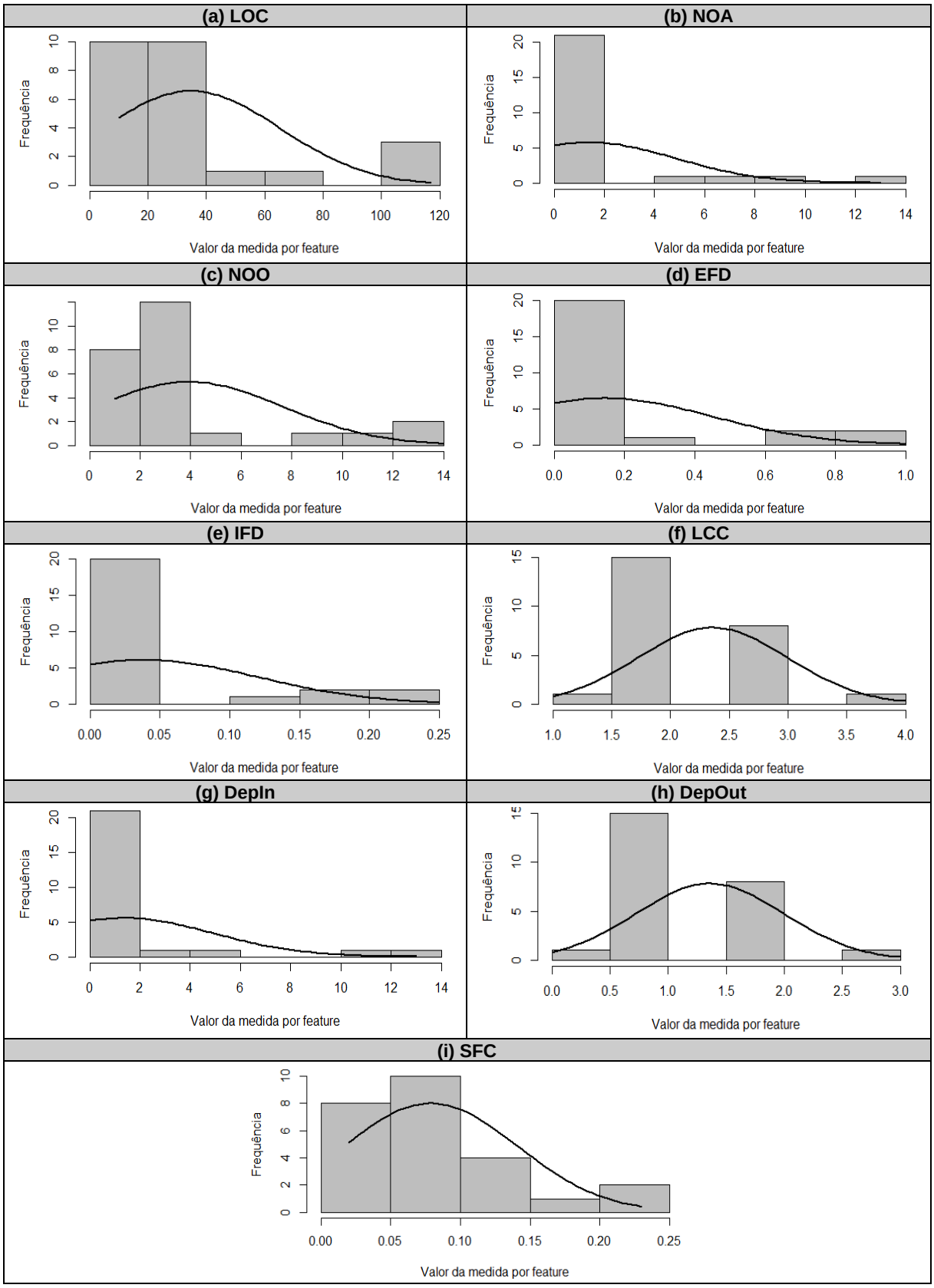


Figura 6.9 - Distribuição dos Dados Amostrais para a Tecnologia Orientação a Aspectos

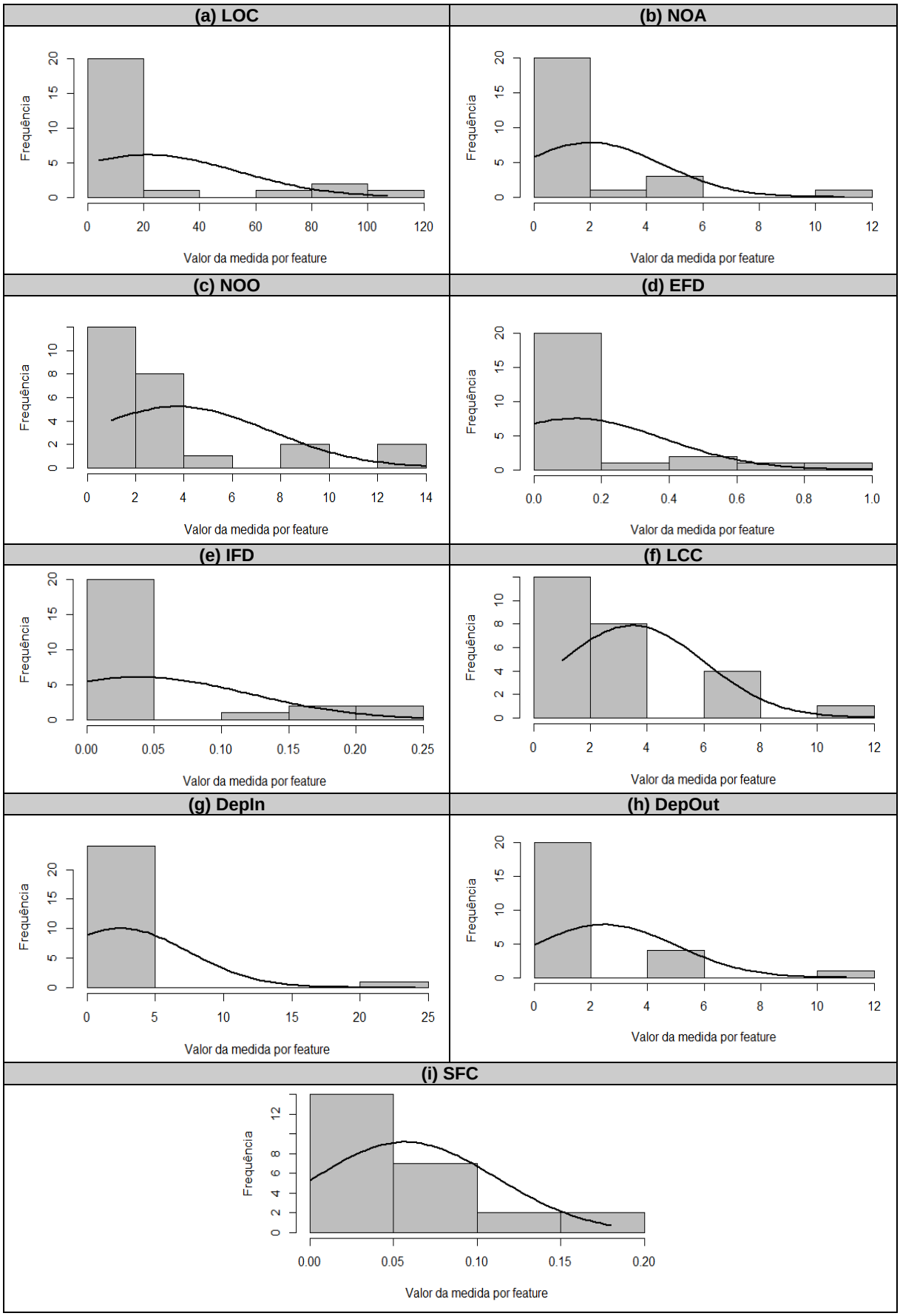
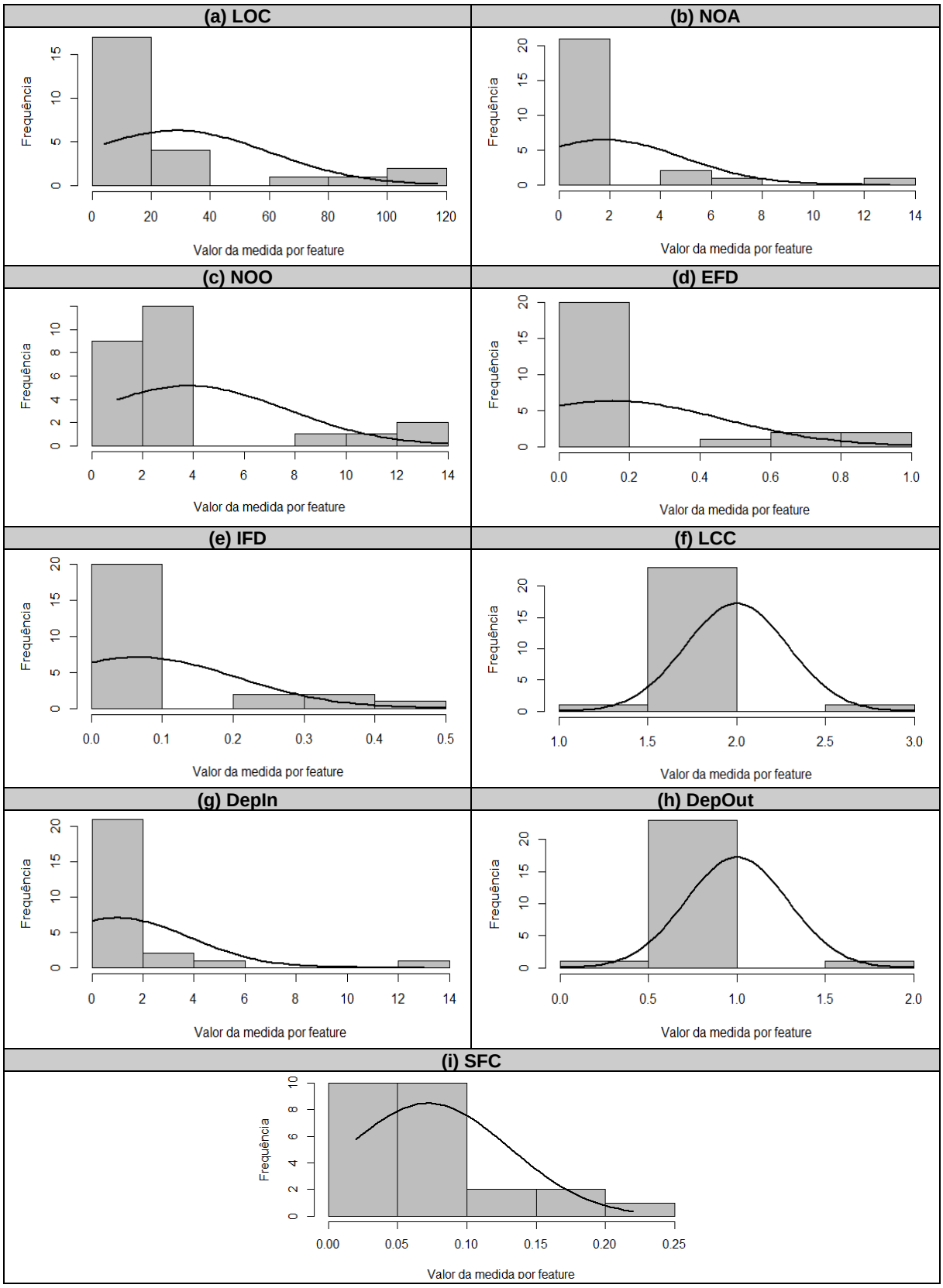


Figura 6.10 - Distribuição dos Dados Amostrais para a Tecnologia Módulos de Características Aspectuais



Posteriormente, foi realizado o tratamento das medidas, uma vez que elas podem assumir valores contínuos ou discretos com diferentes faixas de intervalos. Esse tratamento consistiu na normalização dos dados por meio da Equação 6.4.

$$N = \frac{x - \mu}{\sigma} \tag{6.4}$$

sendo x o valor da medida, μ a média dos dados da amostra e σ o desvio padrão dos dados da amostra. Após essa normalização, foi aplicado o teste de Kolmogorov-Smirnov para verificar se a distribuição dos dados foi alterada. Como resultado, foi identificado que as distribuições continuaram sendo não-normais, exceto para a medida SFC em cada tecnologia. Na Tabela 6.7, são apresentados os valores D_n para a amostra normalizada. Os dados normalizados podem ser encontrados no APÊNDICE D.

Tabela 6.7 - Valores de D_n para o Teste Kolmogorov-Smirnov com os Dados Normalizados

Tecnologias	EFD	IFD	LCC	DepIn	DepOut	SFC
OC	0,47829 2,155 x10^5	0,48191 (1,812 x10^5)	0,3538 (0,003827)	0,48795 (1,352 x10^5)	0,3538 (0,003827)	**0,19625** 0,2907
AO	0,47407 2,635 x10^5	0,48749 (1,383 x10^5)	0,37509 (0,001762)	0,30881 (0,01699)	0,37509 (0,001762)	**0,17334** 0,4403
MCA	0,4828 1,735 x10^5	0,47995 (1,99 x10^5)	0,46 (5,084 x10^5)	0,47748 (2,241x10^5)	0,46 (5,084x10^5)	**0,23357** **0,1307**

Para a escolha do teste de hipóteses, é importante conhecer a distribuição dos dados para escolher o teste mais adequado para avaliar as hipóteses de pesquisa. Dado que a distribuição dos dados é não-normal, exceto para a medida SFC, o teste de hipóteses a ser aplicado deve ser não-paramétrico. Ao utilizar um teste não-paramétrico, é possível trabalhar com diferentes populações; no entanto, para as populações normalmente distribuídas, pode ser mais difícil negar a hipótese nula (VIALI, 2008). Além disso, para a escolha do teste de hipóteses, outras variáveis são consideradas, por exemplo, o nível de mensuração, a quantidade de amostras e se as amostras são dependentes. O nível de mensuração é o ordinal para a qual foi feita a normalização dos valores das medidas para serem contínuos. A quantidade de amostras corresponde a quantidade de tecnologias de gerenciamento de variabilidades, ou seja, têm-se três amostras. Cada tecnologia de gerenciamento de variabilidades implementa o mesmo conjunto de 25 *features*, indicando que existe relação de dependência entre as amostras. Portanto, o teste de significância deve suportar várias amostras dependentes como o teste de Friedman (FRIEDMAN, 1940).

Para o teste de Friedman, os dados são dispostos em uma tabela de dupla entrada com n linhas e k colunas. As linhas representam os conjuntos correspondentes de indivíduos, ou

seja, as *features* da LPS. As colunas representam as diversas amostras, ou seja, as tecnologias de gerenciamento de variabilidades. Assim, o teste determina se as amostras provêm da mesma população, por meio da Equação 6.5 (VIALI, 2008):

$$x^2 = \frac{12}{nk(k+1)} \sum_{i=1}^{k} {R_j}^2 - 3n(k+1) \tag{6.5}$$

sendo `n` a quantidade de *features*, `k` a quantidade de tecnologias de gerenciamento de variabilidades, R_j a soma dos postos da coluna. Os postos são valores discretos atribuídos aos valores de cada linha, os quais são contabilizados em cada amostra de modo a identificar a diferença significativa. Em resumo, a homogeneidade dos postos nas amostras, retornado pelo *p-value*, indica que há diferença entre as amostras.

Para a interpretação do teste de Friedman, foi fixado o nível de 5% de significância. Desse modo, se o *p-value* retornado pelo teste for menor que o nível de significância, indica que existe diferença significativa entre as tecnologias de gerenciamento de variabilidades. Na Tabela 6.8, são apresentados os *p-values* obtidos para as medidas. Como todos os *p-values* retornados foram menores que 0,05, então existe diferença significativa entre as tecnologias de gerenciamento de variabilidades para todas as medidas.

Tabela 6.8 - *P-value* Retornado pelo o Teste de Friedman

	EFD	IFD	LCC	DepIn	DepOut	SFC
p-valor	$4{,}75 \times 10^{-7}$	$1{,}125 \times 10^{-7}$	0,007907	0,0004439	0,007907	0,04243

Ao constatar que existe diferença significativa entre as amostras, as distribuições dos dados das amostras diferem uns dos outros, mas não se sabe entre quais amostras ocorre a diferença. Desse modo, é necessário realizar o *post hoc* para a identificação dos pares de amostras diferentes umas das outras. Assim, foi aplicado o teste de Conover (CONOVER, 1999) com nível de 5% de significância. Da mesma forma, para determinar se existe diferença significativa entre os grupos, o *p-value* retornado deve ser menor que o valor de 0,05. Na Tabela 6.9, é apresentado o *p-value* para o *post hoc*, na qual os valores que indicam diferença significativa são grafados em negrito.

Na Tabela 6.10, é apresentado o agrupamento das tecnologias por diferença significativa. Por exemplo, para a medida EFD, foi constatada diferença significativa entre os grupos OA - OC, MCA - OC e MCA - OA. Desse modo, existem três grupos A, B e C. Para a medida LCC, foi identificado que existe diferença significativa apenas para o grupo MCA - OA; logo, essas tecnologias foram alocadas em grupos diferentes B e A, respectivamente, e a

tecnologia OC, que não possui diferença significativa entre essas tecnologias, foi alocada nos dois grupos AB. Para a medida DepIn, foi constatado que existe diferença significativa entre os grupos MCA - OC e MCA - OA; logo, como existe diferença significativa apenas da tecnologia MCA para as demais, ela foi alocada no grupo B e as demais no grupo A.

Tabela 6.9 - *P-value* Retornado pelo *post hoc* de Conover

EFD		
	OC	OA
OA	**0,0282**	-
MCA	**0,0031**	**$2{,}2\times10^{-6}$**

IFD		
	OC	OA
OA	**0,0068**	-
MCA	**0,0068**	**$8{,}3\times10^{-7}$**

LCC		
	OC	OA
OA	0,1264	-
MCA	0,1264	**0,0031**

DepIn		
	OC	OA
OA	0,77851	-
MCA	**0,00210**	**0,00091**

DepOut		
	OC	OA
OA	0,1264	-
MCA	0,1264	**0,0031**

SFC		
	OC	OA
OA	0,072	-
MCA	0,574	**0,020**

Tabela 6.10 - Agrupamento por Diferença Significativa

EFD		
OC	OA	MCA
A	B	C

IFD		
OC	OA	MCA
A	B	C

LCC		
OC	OA	MCA
AB	A	B

DepIn		
OC	OA	MCA
A	A	B

DepOut		
OC	OA	MCA
AB	A	B

SFC		
OC	OA	MCA
AB	A	B

Ao analisar esses dados, é possível perceber que, para todas as medidas, existe diferença significativa entre as tecnologias OA e MCA. Em relação às medidas de acoplamento (DepIn, DepOut e SFC), não foi constatada diferença significativa entre as tecnologias OA e OC, Excepcionalmente, para a medida SFC, foi constatada diferença entre as tecnologias OC e MCA. Em relação às medidas de coesão (EFD, IFD e LCC), apenas para a medida LCC não foi constatada diferença significativa entre a tecnologia OC e de OA e entre a tecnologia OC e MCA. De modo geral, pode-se dizer que não existe diferença significativa entre a tecnologia OC e as demais tecnologias. No entanto, existe diferença significativa entre OA e MCA.

6.7.2 Análise das Propriedades de Software

Uma vez que as medidas de software foram analisadas separadamente, cabe aplicá-las na Equação 6.1, para a análise da propriedade de acoplamento, e na Equação 6.2, para a análise

da propriedade de coesão, relacionadas à modularidade. Com essas equações, foram obtidos valores agregados das medidas para cada uma das duas propriedades. Os dados obtidos encontram-se documentados no APÊNDICE E. Para descrever e caracterizar a amostra, as medidas de tendência central e as medidas de dispersão são apresentadas para as tecnologias OC, OA e MCA na Tabela 6.11. A partir desses dados, é possível perceber que há pouca ou nenhuma alteração na média e no desvio padrão das medidas de acoplamento e de coesão coletadas em cada tecnologia.

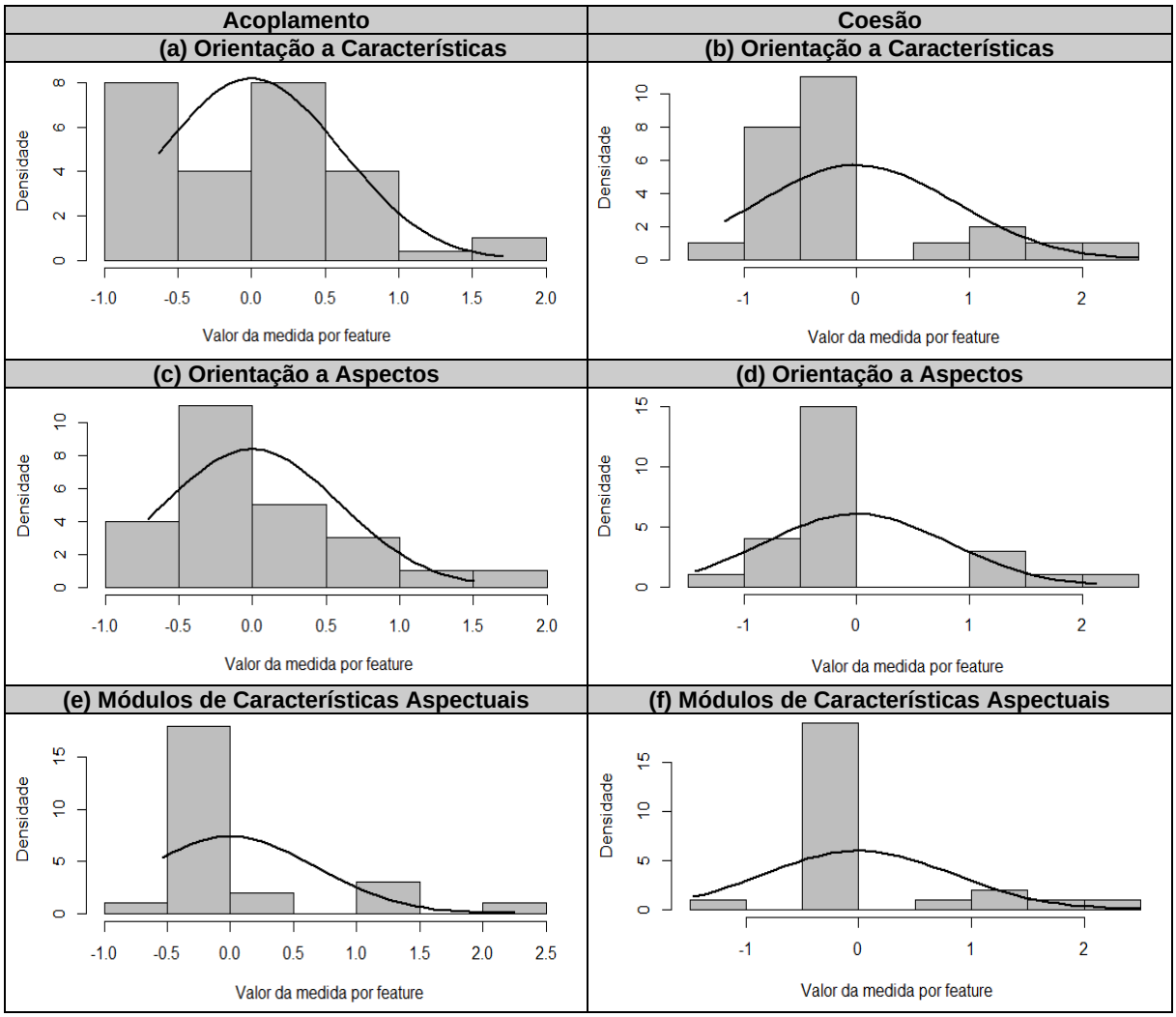
Tabela 6.11 - Estatística Descritiva para a Tecnologia OC

Tecnologia		Mínimo	25%	Mediana	75%	Máximo	Média	Desvio Padrão
OC	**Acoplamento**	-0,63	-0,63	0,11	0,16	1,70	0,00	0,60
	Coesão	-1,17	-0,65	-0,12	-0,12	2,49	0,00	0,88
OA	**Acoplamento**	-0,71	-0,38	-0,34	0,38	1,50	-0.03	0,61
	Coesão	-1,43	-0,25	-0,12	0-,12	2,12	0,00	0,82
MCA	**Acoplamento**	-0,53	-0,41	-0,19	0,12	2,24	0,00	0,67
	Coesão	-1,47	-0,31	-0,31	-0,31	2,49	0,00	0,83

De modo a visualizar a distribuição dos dados, na Figura 6.11, são apresentados histogramas juntamente com a curva de distribuição normal esperada para as propriedades de software. No lado direito, têm-se a distribuição dos dados referentes à propriedade de coesão e, no lado esquerdo, têm-se a distribuição dos dados referentes à propriedade de acoplamento para as tecnologias. Ao analisar os histogramas, apenas as medidas de acoplamento para as tecnologias OC e OA (Figura 6.11(a) e Figura 6.11(c)) não apresentam lacunas em suas distribuições, ou seja, pode ser que a distribuição dos dados seja normal para essa amostra. Diferentemente, os outros histogramas apresentam lacunas, um indicativo da não-normalidade dos dados.

Desse modo, para confirmar a (não) normalidade da distribuição dos dados, foi aplicado o teste de Kolmogorov-Smirnov. Novamente, a interpretação do valor D_n foi feita por meio da tabela de valores críticos de Mille (MILLE, 1956) com o nível de confiança de 5% (`alpha = 5%`) e o tamanho da amostra (`n = 25 features`). Desse modo, para rejeitar a hipótese nula, o valor D_n deve ser maior que o valor crítico que é `0,23768` ou o *p-value* deve ser menor que o intervalo de confiança para as distribuições não serem normais. Os resultados obtidos são apresentados na Tabela 6.12, em que as medidas que possuem uma distribuição normal são grafadas em negrito. Isto é, apenas a propriedade acoplamento apresenta uma distribuição normal para as tecnologias OC e OA.

Figura 6.11 - Distribuição dos Dados Amostrais para as Propriedades de Software



Posteriormente, o teste de Friedman foi aplicado com nível de 5% de significância para verificar se existe diferença significativa entre o conjunto das medidas de acoplamento e das medidas de coesão. O *p-value* obtido para a propriedade de acoplamento e de coesão é apresentado na Tabela 6.13. Como nenhum resultado foi menor que o nível de confiança, conclui-se que não há diferença significativa entre as medidas agregadas das propriedades de acoplamento e de coesão para as tecnologias de gerenciamento de variabilidades, ou seja, não é possível rejeitar a hipótese H_0.

Tabela 6.12 - Valores de D_n para o Teste Kolmogorov-Smirnov com os Dados Normalizados

Tecnologias	Acoplamento	Coesão
OC	**0,16941 (0,4698)**	0,35621 (0,003513)
AO	**0,23516 (0,1259)**	0,3574 (0,003367)
MCA	0,33564 (0,007157)	0,44696 ($9{,}183 \times 10^{5}$)

Tabela 6.13 - *P-value* Retornado pelo Teste de Friedman

	Coesão	Acoplamento
p-valor	0,1023	0,4677

Visto que as tecnologias de gerenciamento de variabilidades comparadas no estudo são baseadas em composição, ou seja, permitem a decomposição modular do sistema em *features*, é compreensível que de modo geral não haja diferença significativa em relação ao acoplamento e à coesão de suas *features*. Contudo, durante a análise das medidas de forma separada, pôde ser identificada a diferença significativa entre as tecnologias OA e MCA, em que esta última apresentou resultados melhores.

6.8 Discussão

A partir da análise das medidas de software agregadas em acoplamento e em coesão, não pôde ser constatada diferença significativa entre a modularidade das tecnologias de gerenciamento de variabilidades para a `LPS WebHelp`. Esse resultado pode ser justificado pela utilização de equações para a agregação das medidas de software nas propriedades de acoplamento e de coesão. Essas equações foram elaboradas considerando que as medidas possuem mesma importância para o cálculo das propriedades de software. No entanto, poderia ter sido investigado, utilizando a análise dos componentes principais (*Prinpipal Component Analysis - PCA*), quais as medidas mais importantes para mensurar cada propriedade de modo a estabelecer seus coeficientes. Além disso, foi considerado um conjunto pequeno de medidas de software, o que pode não mensurar propriedades importantes para a modularidade da LPS.

Contudo, ao observar as medidas de forma individual (não agregadas), foi identificado que há diferença significativa entre as tecnologias OA e MCA para todas as medidas de acoplamento e de coesão. Como decisão de projeto, foi definido que as *features* seriam consideradas a unidade modular de rastreamento entre as tecnologias de gerenciamento de variabilidades. Dessa forma, na LPS implementada na tecnologia OA, os aspectos foram utilizados para o encapsulamento de interesses transversais e para a abstração de características. Porém, essa tecnologia é destinada ao encapsulamento de interesses transversais, ou seja, essa tecnologia lida com características de código compartilhado aplicando extensões homogêneas. Diferentemente, na LPS implementada na tecnologia OC, as classes e os refinamentos foram utilizados para abstrair as características da LPS utilizando extensões heterogêneas. Assim, na LPS implementada na tecnologia MCA, foram mantidas as abstrações desenvolvidas em OC e, posteriormente, foram analisadas as extensões homogêneas e heterogêneas para identificar

onde seriam aplicados os aspectos. Dessa forma, a tecnologia MCA apresentou resultados melhores que a tecnologia OA para as medidas de acoplamento e de coesão.

É interessante destacar que os resultados obtidos a partir da análise das medidas individuais (não agregadas) convergem para o resultado obtido em outro trabalho (GAIA et al., 2012), no qual foi investigada a evolução de LPS utilizando outro conjunto de medidas específicas para o contexto de LPS. Como resultado, identificaram que a tecnologia MCA apresentou resultados melhores que a tecnologia OA para a propagação de mudanças e estabilidade da LPS. Em outras palavras, a LPS desenvolvida em MCA apresenta melhor modularidade que as outras tecnologias, pois a modularidade relaciona-se com a estabilidade da LPS, facilitando a manutenibilidade e a reusabilidade de seus artefatos (COLANZI, 2012).

Além disso, ao não constatar diferença significativa entre as tecnologias OA e OC, os resultados desse trabalho convergem para os obtidos em outro trabalho (REIS et al., 2014), no qual foi investigada a manutenibilidade de uma LPS. Para a investigação da manutenibilidade, foram utilizadas medidas destinadas à OO para mensurar as propriedades de acoplamento, coesão, tamanho e complexidade de software. Como resultado, não constataram diferença significativa entre as tecnologias OA e OC, ao analisar cada medida de software. Contudo, os autores destacam que a tecnologia OA apresenta melhores resultados quando comparada com a tecnologia OC, o que diverge dos resultados obtidos nesse trabalho. Esse desalinhamento pode ser resultante da utilização de medidas de software destinadas à OO. Dessa forma, seriam necessários novos estudos para identificar tendências de utilização entre essas tecnologias.

6.9 Considerações Finais

Nesse capítulo, foram apresentados os processos necessários para o desenvolvimento da LPS. Como o objetivo do trabalho não é a avaliação dos produtos finais derivados da LPS, mas dos artefatos de software reutilizáveis, foi dado enfoque apenas para a Engenharia de Domínio. Inicialmente, foi realizado uma revisão *ad hoc* da literatura por meio da qual foi identificado que a abordagem LPS não é amplamente explorada para o desenvolvimento de sistemas de software de TA (MARTINS et al., 2018).

Diante disso, foi desenvolvida uma LPS para TA visando satisfazer as necessidades de usuários com dislexia. Para isso, foi realizada a Análise de Domínio, em que diversos sistemas que fornecem recursos para pessoas com dislexia foram analisados a fim de identificar seus recursos comuns e variáveis. Posteriormente, foi realizada a Arquitetura de Domínio, em que foram definidos as variantes e os pontos de variabilidades para a realização da modelagem de

domínio da LPS. Em seguida, foi realizada a Implementação de Domínio, que consistiu no desenvolvimento do sistema de software `WebHelpDyslexia` na versão *desktop*, utilizando a linguagem Java. A partir dessa versão, os trechos de código puderam ser extraídos a fim de compor as *features* da `LPS WebHelp`, utilizando três tecnologias de gerenciamento de variabilidades (OA, OC e MCA).

A partir do desenvolvimento da `LPS WebHelp`, utilizando as tecnologias OA, OC e MCA, foi realizada a análise estatística sobre os valores das medidas coletadas em cada versão. Com o agrupamento das medidas nas propriedades de acoplamento e de coesão não pôde ser constatada diferença significativa entre a modularidade da `LPS WebHelp` desenvolvida com essas três tecnologias. Isso pode ser justificado pela agregação das medidas nas propriedades de acoplamento e de coesão, em que foi feita a média aritmética dos valores coletados. Por outro lado, ao analisar os resultados individuais das medidas de acoplamento e de coesão, foi identificado que há diferença significativa entre as tecnologias OA e MCA. Nesse sentido, a `LPS WebHelp` desenvolvida com a tecnologia MCA é menos acoplada e mais coesa do que a desenvolvida com a tecnologia OA. Apesar, desses resultados serem limitados a `LPS WebHelp`, eles convergem para os resultados obtidos em outro trabalho (GAIA et al., 2012). Dessa forma, a tecnologia MCA pode ser mais atrativa para a modularidade de LPS.

7 TRABALHOS RELACIONADOS

7.1 Gerenciamento de Variabilidades

Diversos trabalhos estabelecem comparação entre diferentes tecnologias de gerenciamento de variabilidades. A comparação entre essas tecnologias é realizada para avaliar diferentes características da LPS, para os quais destacam-se os estudos discutidos a seguir.

De modo a investigar a evolução de LPS em termos da propagação de mudanças e estabilidade da LPS, em um trabalho (GAIA et al., 2012), foi realizada uma avaliação quantitativa envolvendo quatro tecnologias de gerenciamento de variabilidades (Compilação Condicional, Orientação a Características, Orientação a Aspectos e Módulos de Características Aspectuais). Para identificar se os Módulos de Características Aspectuais apresentam mais estabilidade e menos impacto relacionado a propagação de mudanças que as outras tecnologias, foi coletado o valor das medidas de software *Concern Diffusion over Components* (CDC), *Concern Diffusion over Operations* (CDO), *Concern Diffusion over Lines of Code* (CDLOC) e *Number of Lines of Concern Code* (LOCC), além de medidas relacionadas a adição, a remoção e a modificação de código da LPS. As medidas foram coletadas a partir da `LPS WebStore`, desenvolvida com propósito acadêmico. Como resultado, foi identificado que a tecnologia de Módulos de Características Aspectuais suporta melhor a propagação de mudanças e apresenta boa estabilidade. Da mesma forma, a Orientação a Características apresenta boa estabilidade. A Compilação Condicional apresentou os piores resultados para propagação de mudanças e estabilidade da LPS.

De modo similar, em outro trabalho (FERREIRA et al., 2014), foi proposta uma investigação na evolução de LPS em termos da propagação de mudanças e da modularidade da LPS. Para isso, utilizaram as tecnologias de gerenciamento de variabilidades Compilação Condicional, Orientação a Objetos e Orientação a Características. O mesmo conjunto de medidas foi coletado, utilizando a `LPS WebStore` e a `LPS MobileMedia`, desenvolvidas com propósito acadêmico. Como resultado, foi identificado que a Orientação a Características requer poucas alterações no código, além de ser mais eficaz no combate a degeneração da modularidade.

Em relação a manutenibilidade em LPS, no terceiro trabalho (REIS et al., 2014), foi investigo se existe diferença significativa entre as tecnologias de gerenciamento de variabilidades Orientação a Aspectos, Orientação a Delta e Orientação a Características. Para isso, foram coletadas medidas relacionadas ao acoplamento, à coesão, à complexidade e ao

tamanho. Especificamente, foram coletadas as medidas no contexto de Orientação a Objetos (*Lines of Code* - LOC, *Lack of Cohesion of Methods* - LCOM, *Depth of Inheritance Tree* - DIT, *Coupling Between Objects* - CBO, *Cyclomatic Complexity* - CC e *Weighted Methods per Class* - WMC). As medidas foram coletadas na `LPS TankWar` desenvolvida com propósito acadêmico. Como resultado, identificaram que não há diferença significativa entre as tecnologias de gerenciamento de variabilidades, para cada medida coletada.

O diferencial deste trabalho consiste no conjunto de medidas de software utilizadas. Para avaliar a modularidade da LPS, foram selecionadas medidas utilizadas no contexto de LPS. Essas medidas são relacionadas às propriedades de acoplamento e de coesão frequentemente utilizadas para avaliar a modularidade de sistemas de software. Nesse sentido, foram coletadas as medidas de acoplamento DepIn, DepOut e SFC e as medidas de coesão LCC, EFD e IFD.

7.2 Utilização de LPS para Desenvolver Sistemas de Tecnologia Assistiva

LPS ainda não foi amplamente explorada para o desenvolvimento de sistemas de software de TA. Para mostrar a sua utilização para o desenvolvimento desses sistemas, destacam-se os trabalhos discutidos a seguir.

Nos trabalhos, um contexto bastante explorado é o de ambientes assistidos. Dada a alta personalização exigida por esses ambientes, em um trabalho (NAKAGAWA et al., 2013), foi verificada a importância de estabelecer um conjunto bem definido de atributos de qualidade para cada contexto de utilização. Frequentemente, os ambientes assistidos são equipados com um conjunto de dispositivos eletrônicos para monitorar e promover a segurança residencial e a qualidade de vida das pessoas com limitações funcionais. Conforme o interesse principal para o uso de ambientes assistidos, podem ser identificados diferentes atributos de qualidade, por exemplo, a confiabilidade, a segurança, a manutenção, a eficiência e a segurança.

Alguns atributos de qualidade foram explorados em outro trabalho (RODRIGUES et al., 2012), no qual é destacado que a má qualidade da disponibilidade, da segurança e da integridade dos sistemas de software pode levar a consequências fatais e de emergência para os idosos que dependem de recursos de ajuda. Ainda, em outro trabalho (PIEPER et al., 2011), foi destacado que esses recursos são utilizados por usuários com limitações funcionais não familiarizados com recursos tecnológicos, o que torna a usabilidade um fator importante a ser considerado. Portanto, os requisitos relacionados ao tipo de mídia, ao *design* e à interação devem ser bem elaborados para tornar o produto utilizável e para facilitar a aceitação do uso.

Além disso, a quantidade de dispositivos que podem ser utilizados em um ambiente assistido torna complexo o processo de desenvolvimento da LPS. Em um trabalho (GAMEZ et al., 2013), foi identificado que, frequentemente, os requisitos dos usuários podem sofrer alterações. Dessa forma, é importante que o modelo de variabilidades evolua para continuar alinhado as mudanças de requisitos. Essa evolução é consequência de mudanças relacionadas às ações de: i) adicionar ou remover recursos; ii) adicionar ou remover grupos de recursos; iii) adicionar ou remover restrições entre recursos; e iv) modificar a variabilidade de um recurso. Essas mudanças devem ser propagadas para os produtos da LPS.

Em outro contexto (CALEFATO, 2015), há destaque para a disponibilização de recursos de TA por ONG's. Dada a limitação de recursos humanos e financeiros, LPS torna-se uma alternativa para o desenvolvimento desses recursos, visto que permite alta customização de produtos e redução nos custos de desenvolvimento. Em especial, foi extraída uma LPS a partir de produtos de software disponibilizados pela ONG italiana *Informatici Senza Frontiere* (ISF). Essa LPS possui recursos voltados à limitação ou à ausência de fala/movimentos. Durante o desenvolvimento, foi identificada a falta de padrões de desenvolvimento para a LPS por causa da falta de experiência dos desenvolvedores. No entanto, foram obtidos ganhos quanto à produtividade e à qualidade dos produtos, além da queda em relação aos custos de desenvolvimento.

É possível perceber que os trabalhos relacionados são limitados quanto ao contexto de utilização que, em maior parte, referem-se a ambientes assistidos. Ainda, esses trabalhos fornecem pouca ou nenhuma informação quanto ao tipo de tecnologia de gerenciamento de variabilidades utilizado; tão pouco, quanto às avaliações de aspectos de qualidade levantados. Nesse sentido, o trabalho proposto visa ao desenvolvimento de uma LPS voltada para recursos de TA para pessoas com dislexia. Ainda, o trabalho diferencia-se pela utilização de diferentes tecnologias para o desenvolvimento da LPS, os quais foram avaliados considerando propriedades relacionadas à modularidade de sistemas de software (coesão e acoplamento).

8 AMEAÇAS A VALIDADE

A validade é uma medida na qual uma conclusão é bem fundamentada de acordo com a concepção e a análise de um experimento (TRAVASSOS et al., 2002). As ameaças a validade são organizadas em quatro tipos:

- **Validade Interna**. Essa validade é a relação observada entre os fatores que afetam o processo de experimentação, sendo o resultado casual. As principais limitações são referentes à seleção de propriedades e de medidas de software para mensurar a modularidade. Dessa forma, as propriedades de software selecionadas foram o acoplamento e a coesão, pois são propriedades frequentemente utilizadas para aferir a modularidade de sistemas de software (BAVOTA et al., 2010; BAVOTA et al., 2012; BECK et al., 2011; COLANZI, 2012). Em relação à seleção de medidas, foi realizado um MSL para identificar quais são as medidas de acoplamento e de coesão frequentemente utilizadas no contexto de LPS. No entanto, apenas seis medidas de software foram selecionadas para a mensuração dessas propriedades. Talvez, se outras medidas forem utilizadas para mensurar essas propriedades, os resultados podem ser diferentes daqueles que foram encontrados;

- **Validade Externa**. Essa ameaça consiste na capacidade de generalizar a descoberta. A coleta e a análise dos dados foram feitas em apenas uma LPS desenvolvida para o domínio de TA. Diversos outros domínios poderiam ser considerados para a coleta dessas medidas, uma vez que existem repositórios de LPS desenvolvidas com finalidades acadêmicas. Porém, para a utilização dessas LPS, seria necessário que elas estivessem disponíveis nas três tecnologias de gerenciamento de variabilidades utilizadas neste trabalho, o que geralmente não ocorre;

- **Validade de Constructo**. Essa ameaça considera se os fatores que afetam o processo de experimentação refletem a causa e o resultado reflete o efeito. A LPS foi desenvolvida utilizando três tecnologias de gerenciamento de variabilidades. As versões da LPS em diferentes tecnologias foram desenvolvidas por apenas um pesquisador. No entanto, esse pesquisador não possuía conhecimento apenas da linguagem Java, sendo necessário um período de aprendizado para utilizar as extensões dessa linguagem (no caso, as linguagens Jakarta e AspectJ). Além disso, o *plug-in* para a coleta das medidas de software também

foi desenvolvido pelo mesmo pesquisador. Porém, esse *plug-in* foi testado e utilizado por outros pesquisadores;

- **Validade de Conclusão**. Essa ameaça está relacionada à capacidade de alcançar uma conclusão correta no que diz respeito à relação entre os fatores que afetam o processo de experimentação e o resultado do experimento. Neste trabalho, são analisadas as propriedades de acoplamento e de coesão ligadas à modularidade da LPS. No entanto, essas propriedades podem não refletir a modularidade da LPS como um todo, pois outras propriedades de software poderiam ser consideradas. Além disso, para agregação das medidas de software em acoplamento e em coesão, poderia ter sido investigado quais as medidas são mais importantes para avaliar essas propriedades, de modo a atribuir coeficientes (pesos) a cada uma delas. Dessa forma, os resultados obtidos por meio da agregação das medidas de software poderiam ser diferentes e, talvez, indicaria diferença significativa entre a modularidade da LPS implementada com as tecnologias de gerenciamento de variabilidades.

9 CONSIDERAÇÕES FINAIS

9.1 Conclusão

Neste trabalho, foi investigado se há diferença significativa entre a modularidade de uma LPS desenvolvida com as tecnologias de gerenciamento de variabilidades: i) Orientação a Características; ii) Orientação a Aspectos; e iii) Módulos de Características Aspectuais. Para a modularização da LPS, as *features* presentes no modelo de características foram consideradas as unidades de decomposição para a rastreabilidade entre as tecnologias de gerenciamento de variabilidades. A partir dessa modularização, é esperado que os artefatos de software possam ser sistematicamente reutilizados para a derivação de produtos da LPS.

A modularidade da LPS foi investigada em termos do acoplamento e da coesão de suas *features*. Para isso, foram desenvolvidas três versões da `LPS WebHelp` utilizando as tecnologias de gerenciamento de variabilidades. Posteriormente, medidas de software foram coletadas em cada versão e agregadas nas propriedades de acoplamento e de coesão utilizando média aritmética. A partir da análise estatística das medidas agregadas não pôde ser constatada diferença significativa entre a modularidade da `LPS WebHelp`. Esse resultado poderia ser diferente caso aplicada a análise dos componentes principais (PCA) para o agrupamento das medidas de software, pois seriam estabelecidos coeficientes para as medidas de modo a expressar sua relevância para as propriedades de software. No entanto, ao observar apenas as medidas individuais (não agregadas) de acoplamento e de coesão de software, foi identificado que há diferença significativa entre as tecnologias Orientação a Aspectos e Módulos de Características Aspectuais para as medidas coletadas.

Nesse sentido, a LPS desenvolvida com a tecnologia Módulos de Características Aspectuais é mais coesa e menos acoplada que a LPS desenvolvida com a tecnologia Orientação a Aspectos. Na Orientação a Aspectos, utiliza-se aspectos para o encapsulamento de interesses transversais. Porém, durante a implementação da `LPS WebHelp`, os aspectos também foram utilizados para modularizar as características da LPS, o que não apresentou ser uma solução adequada. Diferentemente, na tecnologia Módulos de Características Aspectuais, são utilizados módulos de características para a modularização das características da LPS e módulos de características aspectuais para a inserção de aspectos dentro dos módulos de características. Assim, essa tecnologia se mostrou uma solução mais adequada à modularização das características da LPS e ao encapsulamento dos interesses transversais em aspectos.

9.2 Contribuições

A principal contribuição desse trabalho para a área de Engenharia de Software consistiu em investigar se há diferença significativa entre a modularidade de uma LPS desenvolvida com três tecnologias de gerenciamento de variabilidades, em termos do acoplamento e da coesão entre suas *features*. Para isso, foi necessário identificar as medidas de software frequentemente utilizadas para mensurar as propriedades de acoplamento e de coesão em LPS. Essas medidas foram implementadas em um *plug-in* para o Eclipse de modo a apoiar a mensuração da LPS. Além disso, a LPS desenvolvida contempla o domínio de TA, contribuindo para essa área ao apresentar a viabilidade de utilização de LPS para o desenvolvimento de sistemas de TA e apresentar a realização das atividades da Engenharia de Domínio.

Em resumo, as contribuições do trabalho são:

- A realização de um MSL sobre as características e subcaracterísticas de qualidade de produtos de software investigadas no contexto do LPS, bem como as propriedades e as medidas de software utilizadas para sua mensuração;
- A realização de uma revisão *ad hoc* da literatura de modo a identificar como a abordagem LPS tem sido utilizada para o desenvolvimento de sistemas de TA e como é realizada a avaliação da qualidade dessas LPS;
- O desenvolvimento de um *plug-in* para o Eclipse para a mensuração da modularidade de LPS implementadas com as linguagens de programação AspectJ e Jakarta. Para a mensuração da modularidade são consideradas medidas de acoplamento (DepIn, DepOut e SFC) e de coesão (EFD, IFD e LCC), frequentemente utilizadas no contexto de LPS;
- O desenvolvimento de uma LPS para recursos de TA, em especial, para pessoas com dislexia. A partir do desenvolvimento dessa LPS, as atividades referentes à Engenharia de Domínio puderam ser demonstradas para a criação dos artefatos de software da LPS, utilizando diferentes tecnologias de gerenciamento de variabilidades.
- A identificação da diferença significativa entre a modularidade da LPS desenvolvida com as tecnologias Orientação a Aspectos e Módulos de Características Aspectuais, durante a análise das medidas de acoplamento e de coesão (não agregadas).

9.3 Publicações

- COUTO, M. S. C.; MARTINS, L. A.; COSTA, H.; TERRA, R. MCL: Metrics-based Constraint Language. In **Proceedings of the XIV Brazilian Symposium on Information Systems (SBSI'18).** ACM, New York, NY, USA, Article 9, 8 pages. 2018.
- MARTINS, L. A.; FREIRE, A. P.; PARREIRA JÚNIOR, P. A.; COSTA, H. Sistemas de Tecnologia Assistiva. In: **I Jornada Latino-Americana de Atualização em Informática**. 1 ed. Porto Alegre: Sociedade Brasileira de Computação, p. 212-235. 2018.
- MARTINS, L. A.; FREIRE, A. P.; PARREIRA JÚNIOR, P. A.; COSTA, H. Analysis of Usability Practices in a Project of a Free Assistive Technology System. In **Proceedings of the XIV Brazilian Symposium on Information Systems (SBSI'18).** ACM, New York, NY, USA, n 65, 8 p. 2018.
- MARTINS, L. A.; FREIRE, A. P.; PARREIRA JÚNIOR, P. A.; COSTA, H. Exploratory Study on the Use of Software Product Lines in the Development of Quality Assistive Technology Software. In **Proceedings of the 8th International Conference on Software Development and Technologies for Enhancing Accessibility and Fighting Info-exclusion (DSAI 2018).** ACM, New York, NY, USA, p. 262-269. 2018.
- MARTINS, L. A.; FREIRE, A. P.; PARREIRA JÚNIOR, P. A.; COSTA, H. SPLiME: A tool for Software Product Lines Maintainability Evaluation. In: **X Brazilian Conference on Software: Theory and Practice**, Salvador. Sessão de ferramentas do CBSoft, 2019.

9.4 Trabalhos Futuros

Como trabalhos futuros, sugere-se:

- Investigar formas para a derivação de medidas, de modo a identificar quais as propriedades de software mais relevantes para a mensuração da modularidade;
- Realizar estudos empíricos com diferentes métodos para abordar outras subcaracterísticas de Manutenibilidade, de forma a avançar na compreensão de formas de mensuração do construto teórico de maneira mais aprofundada.
- Melhorar a avaliação do estudo por meio de sua aplicação em um conjunto com LPS de diversos domínios para generalizar os resultados obtidos.

REFERÊNCIAS

APEL, S.; BATORY, D.; KÄSTNER, C.; SAAKE, G. **Feature-Oriented Software Product Lines.** 1. ed. New York, NY, USA: Springer Science & Business Media. 315 p. 2013.

APEL, S.; BEYER, D. Feature Cohesion in Software Product Lines: An Exploratory Study. In: **International Conference on Software Engineering (ICSE)**, IEEE Computer Society. p. 421-430. 2011.

APEL, S.; LEICH, T.; SAAKE, G. Aspectual Feature Modules. In: **IEEE Transactions on Software Engineering**, IEEE Computer Society, v. 34, n. 2, p. 162-180, 2008.

ARCAINI, P.; GARGANTINI, A.; VAVASSORI, P. Automated Repairing of Variability Models. In: **International Systems and Software Product Line Conference**, New York, NY, USA. Volume A (SPLC'17), p. 9-18. 2017.

ASPECTJ TEAM. **The AspectJ Programming Guide.** 2008. Disponível em: <https://www.eclipse.org/aspectj/doc/released/progguide/>. Acesso em: 28 de novembro de 2017.

AVELAR, L. O.; REZENDE, G. C.; FREIRE, A. P. WebHelpDyslexia: A Browser Extension to Adapt Web Content for People with Dyslexia. In**: Procedia Computer Science**, 67. p. 150-159. 2015.

BABAR, M. A.; CHEN, L.; SHULL, F. Managing Variability in Software Product Lines. In: **IEEE Software, IEEE Computer Society**, v. 27, n. 3, p. 89-91, 2010.

BAGHERI, E.; GASEVIC, D. Assessing the Maintainability of Software Product Line Feature Models Using Structural Metrics. In**: Software Quality Journal**, Springer US, v. 19, n. 3, p. 579-612, 2011.

BANSIYA, J.; DAVIS, C. G. A Hierarchical Model for Object-Oriented Design Quality Assessment. In: **IEEE Transactions on Software Engineering**, v. 28, n. 1, p. 4-17, 2002.

BATORY, D.; SARVELA, J. N.; RAUSCHMAYER, A. Scaling Step-Wise Refinement. In: **IEEE Transactions on Software Engineering**, v. 30, n. 6, p. 355-371, 2004.

BAVOTA, G.; DE LUCIA, A.; Marcus, A.; Oliveto, R. Software Re-Modularization Based on Structural and Semantic Metrics. In: **Working Conference on Reverse Engineering** (pp. 195-204). 2010.

BAVOTA, G.; DE LUCIA, A.; MARCUS, A.; OLIVETO, R. Using Structural and Semantic Measures to Improve Software Modularization. In: **Empirical Software Engineering**, 18(5), 901-932. 2013.

BECK, F.; DIEHL, S. On the Congruence of Modularity and Code Coupling. In: **European Conference on Foundations of Software Engineering**. p. 354-364. 2011.

BENDUHN, F. Contract-Aware Feature Composition. Monografia (Bachelor's thesis) - Otto Von Guericke Universität Magdeburg - Computer Science, Magdeburg, 109 p., 2012.

BOUCHER, Q.; CLASSEN, A.; FABER, P.; HEYMANS, P. Introducing TVL, a Text-Based Feature Modelling Language. In**: International Workshop on Variability Modelling of Software-Intensive Systems**, Linz, Austria, January. p. 27-29. 2010.

CALEFATO, F.; NICOLÒ, R. D.; LANUBILE, F.; LIPPOLIS, F. Product Line Engineering for NGO Projects. In: **International Workshop on Product Line Approaches in Software Engineering**, Piscataway, NJ, USA: IEEE Computer Society. p. 3-6. 2015.

CASTELLUCIA, D.; BOFFOLI, N. Service-Oriented Product Lines: A Systematic Mapping Study. In: **ACM SIGSOFT Software Engineering Notes**, ACM, v. 39, n. 2, p. 1-6, 2014.

CGEE. CENTRO DE GESTÃO E ESTUDOS ESTRATÉGICOS. **Mapeamento de Competências em Tecnologia Assistiva**. Brasília, 2012. 381 p. Disponível em: <https://www.cgee.org.br/ relatorios>. Acesso em: 09 de janeiro de 2018.

CHIDAMBER, S. R.; KEMERER, C. F. A Metrics Suite for Object Oriented Design. In**: IEEE Transactions on Software Engineering**, v. 20, n. 6, p. 476-493, 1994.

CIRILO, E. GENARCH: Uma Ferramenta Baseada em Modelos para Derivação de Produtos de Software. 100 p. Dissertação (Master's thesis) - Pontifícia Universidade Católica do Rio de Janeiro (PUC-RIO), Rio de Janeiro, 2008.

COLANZI, T. E.; VERGILIO, S. R.; GIMENES, I. M. S.; OIZUMI, W. N. A Search-Based Approach for Software Product Line Design. In**: International Software Product Line Conference.** V. 1. p. 237-241. 2014.

COLANZI, Thelma Elita. Search based design of software product lines architectures. In: **Proceedings of the 34th International Conference on Software Engineering**. IEEE Press, 2012. p. 1507-1510.

CONOVER, W. **Practical Nonparametric Statistics**. 3. ed. John Wiley & Sons. INC, New York, 592 p. 1999.

CÔTÉ, M.-A.; SURYN, W.; GEORGIADOU, E. In Search for a Widely Applicable and Accepted Software Quality Model for Software Quality Engineering. In: **Software Quality Journal**, Springer US, v. 15, n. 4, p. 401-416, 2007.

COUTO, M. V. de Á. Extração de Linhas de Produtos de Software: Um Estudo de Caso Usando Compilação Condicional. Dissertação (Master's Thesis) - Pontifícia Universidade Católica de Minas Gerais, Belo Horizonte. 2010.

CZARNECKI, K.; EISENECKER, U. **Generative Programming: Methods, Tools, and Applications.** 6. ed. New York, NY, USA: Addison Wesley Reading. 832 p. 2000.

CZARNECKI, K.; HELSEN, S.; EISENECKER, U.W. Formalizing cardinality-based feature models and their specialization. In: **Software Process: Improvement and Practice**, 10(1):7-29, 2005.

DEACON, J. **Model-View-Controller (MVC) Architecture**. Acessado em: 10 de março de 2006. http://www. jdl. co. uk/briefings/MVC. pdf, 2009.

DINIZ, J. P.; VALE, G.; GAIA, F.; FIGUEIREDO, E. Evaluating Delta-Oriented Programming for Evolving Software Product Lines. In: **International Workshop on Variability and Complexity in Software Design**, IEEE Press, p. 27-33, 2017

FERREIRA, G. C. S.; GAIA, F. N.; FIGUEIREDO, E.; MAIA, M. DE A. On the Use of Feature-Oriented Programming for Evolving Software Product Lines - A Comparative Study. In**: Science of Computer Programming**, v. 93, p. 65 - 85, 2014. Special Issue with Selected Papers from the Brazilian Symposium on Programming Languages (SBLP 2011).

FERREIRA, G.; MALIK, M.; KÄSTNER, C.; PFEFFER, J.; APEL, S. Do #ifdefs Influence the Occurrence of Vulnerabilities? An Empirical Study of the Linux Kernel. In: **International Systems and Software Product Line Conference**. New York, NY, USA: ACM. p. 65-73. 2016.

Friedman, M. A Comparison of Alternative Tests of Significance for the Problem of m Rankings. In: **The Annals of Mathematical Statistics**, 11(1):86 - 92, 1940.

GAIA, F. N. Uma Avaliação Quantitativa de Módulos de Características Aspectuais para Evolução de Linhas de Produtos de Software. 77 p. Dissertação (Master's thesis) - Universidade Federal de Uberlândia (UFU) - Programa de Pós Graduação em Ciência da Computação, Uberlândia, MG, 2013.

GAIA, F. N.; FERREIRA, G. C. S.; FIGUEIREDO, E.; MAIA, M. DE A. A Quantitative Assessment of Aspectual Feature Modules for Evolving Software Product Lines. In: **Brazilian Symposium on Programming Languages**. Springer, Berlin, Heidelberg, p. 134-149, 2012.

GAMEZ, N.; FUENTES, L. Architectural Evolution of FamiWare Using Cardinality-Based Feature Models. In: **Information and Software Technology**, Elsevier Science Inc., v. 55, n. 3, p. 563-580, 2013.

GIL, R. C.; PIVETA, E. K.; SACCOL, D. D. B.; FAVERI, C. de. A Tool for Searching in Unstructured Code Aspectj. In: Brazilian Symposium on Information Systems: In: **Information Systems: A Computer Socio-Technical Perspective**. Porto Alegre, Brazil, Brazil: Brazilian Computer Society, v.1, p. 6., 2015.

GRISS, M.; FAVARO, J.; D'ALESSANDRO, M. Integrating feature modeling with the RSEB. In **Proceedings of the Fifth International Conference on Software Reuse**, IEEE, pages 76-85, Canada, 1998.

GOVERNO FEDERAL. eMAG - **Modelo de Acessibilidade do Governo Eletrônico**. 2014. Disponível em: <http://emag.governoeletronico.gov.br/cursodesenvolvedor/>. Acessado em: 23 de julho de 2019.

HER, J. S.; KIM, J. H.; OH, S. H.; RHEW, S. Y.; KIM, S. D. A Framework for Evaluating Reusability of Core Asset in Product Line Engineering. In: **Information and Software Technology**, Elsevier, v. 49, n. 7, p. 740-760, 2007.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION AND INTERNATIONAL ELECTROTECHNICAL COMMISSION. **ISO/IEC 25010: Systems and Software Engineering - Systems and Software Quality Requirements and Evaluation.** Switzerland, 2011.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION AND INTERNATIONAL ELECTROTECHNICAL COMMISSION. **ISO/IEC 33000: Information Technology - Process Assessment**. Switzerland, 2015.

JAVED, M.; NAEEM, M.; UMAR, A. I.; BAHADUR, F. Automated Inconsistency Detection in Feature Models: A Generative Programming Based Approach. In: **Open Access Journals**, v. 3, n. 2, p. 59-74, 2016.

JUNG, C. F. Metodologia aplicada a projetos de pesquisa: Sistemas de Informação & Ciência da Computação. **Proposta de TCC e Projeto de Pesquisa**, 2009.

KANG, K. C.; COHEN, S. G.; HESS, J. A.; NOVAK, W. E.; PETERSON, A. S. **Feature-Oriented Domain Analysis (FODA) Feasibility Study**. Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst, 1990.

KANG, K. C.; SUGUMARAN, V.; PARK, S. **Applied Software Product Line Engineering**. 1. ed. New York, NY, USA: CRC press. 561p. 2009.

KÄSTNER, C.; APEL S.; ROSENTHAL, M.; DREILING, A. CIDE: **Virtual Separation of Concerns.** 2010. Disponível em: <http://ckaestne.github.io/CIDE/>. Acesso em: 20 de novembro de 2017.

KÄSTNER, C.; APEL, S. Integrating Compositional and Annotative Approaches for Product Line Engineering. In: **Workshop on Modularization, Composition, and Generative Techniques for Product Line Engineering**. Nashville, TN, USA: University of Passau. p. 35-40. 2008.

KÄSTNER, C.; APEL, S.; KUHLEMANN, M. Granularity in Software Product Lines. In: **International Conference on Software Engineering**. New York, NY, USA. p. 311-320. 2008.

LIEBIG, J.; APEL, S.; LENGAUER, C.; KÄSTNER, C.; SCHULZE, M. An Analysis of the Variability in Forty Preprocessor-Based Software Product Lines. In: **ACM/IEEE International Conference on Software Engineering**. New York, NY, USA. V. 1. p. 105-114. 2010.

LILLIEFORS, H. W. On the Kolmogorov-Smirnov Test for Normality with Mean and Variance Unknown. **Journal of the American Statistical Association**, 62(318):399 - 402, 1967.

MARTIN, R. OO Design Quality Metrics - An Analysis of Dependencies. In: **Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA).** p. 151-170. 1994.

MARTINS, L.; PARREIRA JÚNIOR, P.; FREIRE, A.; COSTA, H. Exploratory Study on the Use of Software Product Lines in the Development of Quality Assistive Technology Software. In: **International Conference on Software Development and Technologies for Enhancing Accessibility and Fighting Info-exclusion.** ACM, New York, NY, USA, 262-269. 2018.

MILLER, L. H. Table of percentage points of kolmogorov statistics. In **Journal of the American Statistical Association**, 51(273):111–121, 1956.

MONTALVILLO, L.; DÍAZ, O. Requirement-Driven Evolution in Software Product Lines: A Systematic Mapping Study. In: **Journal of Systems and Software**, Elsevier Science Inc., v. 122, p. 110-143, 2016.

NAKAGAWA, E. Y.; ANTONINO, P. O.; BECKER, M.; MALDONADO, J. C.; STORF, H.; VILLELA, K. B.; ROMBACH, D. Relevance and Perspectives of AAL in Brazil. In: **Systems and Software**, Elsevier Science Inc., v. 86, n. 4, p. 985-996, 2013.

NORTHROP, L; CLEMENTS, P. **A Framework for Software Product Line Practice**. 2012. Disponível em: <https://resources.sei.cmu.edu/asset_files/WhitePaper/ 2012_019_001_ 495381.pdf>. Acesso em: 05 de novembro de 2017.

PESSOA, L.; FERNANDES, P.; CASTRO, T.; ALVES, V.; RODRIGUES, G. N.; CARVALHO, H. Building Reliable and Maintainable Dynamic Software Product Lines: An Investigation in the Body Sensor Network Domain. In: **Information and Software Technology**, Elsevier Science Inc., v. 86, p. 54-70, 2017.

PIEPER, M.; ANTONA, M.; CORTÉS, U. Ambient Assisted Living. In: **Ercim News**, v. 87, p. 18-19, 2011.

POHL, K.; BÖCKLE, G.; LINDEN, F. J. van D. **Software Product Line Engineering: Foundations, Principles and Techniques**. 1. ed. Berlin, Heidelberg, New York: Springer. 490 p. 2005.

PRODANOV, C. C.; FREITAS, E. C. de. **Metodologia do Trabalho Científico: Métodos e Técnicas da Pesquisa e do Trabalho Acadêmico**. 2. ed. Novo Hamburgo, RS, BR: Editora Feevale. 276 p. 2013.

QUEIROZ, R.; PASSOS, L.; VALENTE, M. T.; HUNSEN, C.; APEL, S.; CZARNECKI, K. The Shape of Feature Code: An Analysis of Twenty C-Preprocessor-Based Systems. In: **Software & Systems Modeling**, Springer, v. 16, n. 1, p. 77-96, 2017.

REIS, J. N.; VALE, G.; COSTA, H. Manutenibilidade de Tecnologias para Programação de Linhas de Produtos de Software: Um Estudo Comparativo. In: **Brazilian Symposium on Software Quality**. 8 pages, 2014.

REVELLE, M.; GETHERS, M.; POSHYVANYK, D. Using Structural and Textual Information to Capture Feature Coupling in Object-Oriented Software. In: **Empirical Software Engineerin**g, v. 16, n. 6, p. 773-811. 2011.

RODRIGUES, G. N.; ALVES, V.; SILVEIRA, R.; LARANJEIRA, L. A. Dependability Analysis in the Ambient Assisted Living Domain: An Exploratory Case Study. In: **Journal of Systems and Software**, v. 85, n. 1, p. 112-131, 2012.

SANTOS, M. C. B.; COLANZI, T. E.; AMARAL, A. M. M. M.; OLIVEIRA JÚNIOR, E. Preliminary Study on the Correlation of Objective Functions to Optimize Product-Line Architectures. In: **Brazilian Symposium on Software Components, Architectures, and Reuse**. p. 11:1-11:10, 2017.

SEGURA, S.; HIERONS, R. M.; BENAVIDES, D.; RUIZ-CORTES, A. Automated Test Data Generation on the Analyses of Feature Models: A Metamorphic Testing Approach. In:

International Conference on Software Testing, Verification and Validation. Washington, DC, USA. p. 35-44. 2010.

SILVA, F.; SILVEIRA, P. A.; GARCIA, V.; E MUNIZ, P. **Linhas de Produtos de Software: Uma tendência da indústria.** Sociedade Brasileira de Computação, 2011. Disponível em:<https://www.researchgate.net/publication/236784308_Linhas_de_Produtos_de_Software_Uma_tendencia_da_industria> Acessado em: 08 de Julho de 2019.

Sobernig, S.; Apel, S.; Kolesnikov, S.; Siegmund, N. Quantifying Structural Attributes of System Decompositions in 28 Feature-Oriented Software Product Lines. In: **Empirical Software Engineering**, Springer US, v. 21, n. 4, p. 1670-1705, 2016.

SOFTEX, S. Modelo MPS - **Melhoria de Processo do Software Brasileiro**, Guia Geral, Guia de Aquisição, Guia de Implementação, Guia de Avaliação. 2009. Disponível em: <http://www.softex.br/mpsbr>. Acesso em: 07 de maio de 2018.

SOMMERVILLE, I. **Software Engineering**. 10th. ed. Pearson Education, 2016. 816 p.

TEAM, CMMI Product. CMMI for Development, Version 1.3, Improving Processes for Developing Better Products and Services. no. CMU/SEI-2010-TR-033. Software Engineering Institute, 2010.

THÜM, T.; APEL, S.; KÄSTNER, C.; SCHAEFER, I.; SAAKE, G. A Classification and Survey of Analysis Strategies for Software Product Lines. In: **ACM Computing Surveys**, v. 47, n. 1, p. 6, 2014.

THÜM, T.; KÄSTNER, C.; BENDUHN, F.; MEINICKE, J.; SAAKE, G.; LEICH, T. FeatureIDE: An Extensible Framework for Feature-Oriented Software Development. In: **Science of Computer Programming**, Elsevier Science Inc., v. 79, p. 70-85, 2014.

VALE, G. A. D. **Avaliação da Manutenibilidade de Sistemas de Software Derivados de Linhas de Produtos de Software**. Graduation's Monograph. Universidade Federal de Lavras, Lavras, MG, 2013.

van GURP, J.; BOSCH, J.; SVAHNBERG, M. On the notion of variability in software product lines. In **Proceedings of the Working IEEE/IFIP Conference on Software Architecture (WICSA'01)**, IEEE Computer Society, pages 45-54, 2001.

VIALI, L**. Análise Estatística Não Paramétrica**.43 p. Porto Alegre, 2008. Disponível em <http://www.mat.ufrgs.br/~viali/estatistica/mat2282/material/apostilas/Testes_Nao_Parametricos.pdf>. Acesso em: 25 de julho de 2019.

WIERINGA N.; MAIDEN, N. M. R.; ROLLAND, C. Requirements Engineering Paper Classification and Evaluation Criteria: A Proposal and a Discussion. Requirements engineering, v. 11, n. 1, p. 102-107, 2006.

APÊNDICE A - ARTIGOS SELECIONADOS

Tabela Apêndice A-1 - Artigos Selecionados

ID	Título	Autores	Ano	Biblioteca	Solução	Atividade SPL	Metodologia
P1	A Case Study of Software Product Line for Business Applications Changeability Prediction	Roško, Z. Strahonja, V.	2014	Scopus	Medida	Implementação de Domínio	Pesquisa de Avaliação
P2	A Comparison of Test Case Prioritization Criteria for Software Product Lines	Sanchez, A. B. Segura, S. Ruiz-Cortes, A.	2014	IEEE	Método	Teste de Produto	Pesquisa de Validação
P3	A Framework for Evaluating Reusability of Core Asset in Product Line Engineering	Her, J. S. Kim, J. H. Oh, S. H. Rhew, S. Y. Kim, S. D.	2007	Science	Modelo	Arquitetura de Domínio	Pesquisa de Avaliação
P4	A Method to Derive Metric Thresholds for Software Product Lines	Vale, G. Figueiredo, E.	2015	IEEE	Método, Ferramenta	Implementação de Domínio	Pesquisa de Validação
P5	A Quantitative and Qualitative Assessment of Aspectual Feature Modules for Evolving Software Product Lines	Gaia, F. Ferreira, G. Figueiredo, E. Maia, M.	2014	Scopus	Medida	Implementação de Domínio	Pesquisa de Avaliação
P6	A Search-Based Approach for Software Product Line Design	Colanzi, T. Vergilio, S. Gimenes, I. Oizumi, W.	2014	ACM	Método	Arquitetura de Domínio	Pesquisa de Validação
P7	Aggregating Measures Using Fuzzy Logic for Evaluating Feature Models	Bezerra, C. I. M. Andrade, R. M. C. Monteiro, J. M. S. Cedraz, D.	2018	ACM	Medida	Arquitetura de Domínio	Pesquisa de Avaliação
P8	An Analysis of the Variability in Forty Preprocessor-Based Software Product Lines	Liebig, J. Apel, S. Lengauer, C. Kästner, C. Schulze, M.	2010	Scopus	Medida, Ferramenta	Implementação de Domínio	Pesquisa de Avaliação
P9	An Empirical Study of Pre-Release Software Faults in an Industrial Product Line	Devine, T. R. Goseva-Popstajanova, K. Krishnan, S. Lutz, R. R. Li, J. J.	2012	IEEE	Medida	Teste de Produto	Pesquisa de Avaliação

Tabela Apêndice A-1 - Artigos Selecionados (cont.)

ID	Título	Autores	Ano	Biblioteca	Solução	Atividade SPL	Metodologia
P10	An Exploratory Study of the Design Impact of Language *Features* for Aspect-Oriented Interfaces	Dyer, R. Rajan, H. Cai, Y.	2012	ACM	Medida	Implementação de Domínio	Pesquisa de Avaliação
P11	An Index-Based Method to Manage the Tradeoff Between Diversity and Commonality During Product Family Design	Thevenot, H. J. Alizon, F. Simpson, T. W. Shooter, S.B.	2007	Scopus	Método	Análise de Domínios	Proposta de Solução
P12	An Integrated Approach to Product Family Redesign Using Commonality and Variety Metrics	Jung, S. Simpson, T. W.	2015	Scopus	Método	Arquitetura de Domínio	Pesquisa de Avaliação
P13	Analyzing the Feature Models Maintainability Over Their Evolution Process: An Exploratory Study	Bezerra, C. I. M. Monteiro, J. M. Andrade, R. M. C. Rocha, L. S.	2016	ACM	Medida, Ferramenta	Arquitetura de Domínio	Pesquisa de Avaliação
P14	Are Change Metrics Good Predictors for an Evolving Software Product Line?	Krishnan, S. Strasburg, C. Lutz, R. R. Goševa-Popstojanova, K.	2011	ACM	Medida	Implementação de Produto	Pesquisa de Avaliação
P15	AspectJ-Based Idioms for Flexible Feature Binding	Andrade, R. Rebêlo, H. Ribeiro, M. Borba, P.	2013	Scopus	Método	Implementação de Domínio	Pesquisa de Avaliação
P16	Assessing Idioms for a Flexible Feature Binding Time	Andrade, R. Ribeiro, M. Rebêlo, H. Borba, P. Gasiunas, V. Satabin, L.	2015	Scopus	Método	Implementação de Domínio	Pesquisa de Avaliação
P17	Assessing Idioms for Implementing Features with Flexible Binding Times	Andrade, R. Ribeiro, M. Gasiunas, V. Satabin, L. Rebêlo, H. Borba, P.	2011	IEEE	Método	Implementação de Domínio	Pesquisa de Avaliação
P18	Assessing the Maintainability of Software Product Line Feature Models Using Structural Metrics	Bagheri, E. Gasevic, D.	2011	Scopus	Medida, Ferramenta	Teste de Domínio	Pesquisa de Validação
P19	Assessment and Cross-Product Prediction of Software Product Line Quality: Accounting for Reuse across Products, over Multiple Releases	Devine, T. Goseva-Popstojanova, K. Krishnan, S. Lutz, R.R.	2016	Scopus	Medida	Implementação de Produto	Pesquisa de Avaliação

Tabela Apêndice A-1 - Artigos Selecionados (cont.)

ID	Título	Autores	Ano	Biblioteca	Solução	Atividade SPL	Metodologia
P20	Assessment of the Design Modularity and Stability of Multi-Agent System Product Lines	Nunes, C. Kulesza, U. Sant'Anna, C. Nunes, I. Garcia, A. Lucena, C.	2009	Scopus	Medida	Implementação de Produto	Pesquisa de Avaliação
P21	Cost-Effective Test Suite Minimization in Product Lines Using Search Techniques	Wang, S. Ali, S. Gotlieb, A.	2015	Science	Medida	Teste de Produto	Pesquisa de Avaliação
P22	Defining and Applying Detection Strategies for Aspect-Oriented Code Smells	Macia, I. Garcia, A. Von Staa, A.	2010	Scopus	Medida	Implementação de Domínio	Pesquisa de Avaliação
P23	Defining Metric Thresholds for Software Product Lines: A Comparative Study	Vale, G. Albuquerque, D. Figueiredo, E. Garcia, A.	2015	ACM	Método	Implementação de Domínio	Pesquisa de Validação
P24	Detecting Code Smells in Software Product Lines: An Exploratory Study	Abilio, R. Padilha, J. Figueiredo, E. Costa, H.	2015	Scopus	Medida	Implementação de Domínio	Pesquisa de Validação
P25	DyMMer: A Measurement-Based Tool to Support Quality Evaluation of DSPL Feature Models	Bezerra, C. Barbosa, J. Freires, J. Andrade, R. Monteiro, J.	2016	ACM	Medida, Ferramenta	Arquitetura de Domínio	Proposta de Solução
P26	Empirical Validation of Complexity and Extensibility Metrics for Software Product Line Architectures	Oliveira Jr., E. A. Maldonado, J. C. Gimenes, I. M. S.	2010	IEEE	Medida	Arquitetura de Domínio	Pesquisa de Validação
P27	Evaluating Delta-Oriented Programming for Evolving Software Product Lines	Diniz, J. P. Vale, G. Gaia, F. Figueiredo, E.	2017	Scopus	Medida	Implementação de Produto	Pesquisa de Avaliação
P28	Evaluating Scenario-Based SPL Requirements Approaches: The Case for Modularity, Stability and Expressiveness	Alférez, M. Bonifácio, R. Teixeira, L. Accioly, P. Kulesza, U. Moreira, A. Araújo, J. Borba P.	2014	Scopus	Medida	Implementação de Produto	Pesquisa de Avaliação

Tabela Apêndice A-1 - Artigos Selecionados (cont.)

ID	Título	Autores	Ano	Biblioteca	Solução	Atividade SPL	Metodologia
P29	Exploring Quality Measures for the Evaluation of Feature Models: A Case Study	Bezerra, C. Andrade, R. Monteiro, J.	2017	Science	Medida	Arquitetura de Domínio	Evaluation reserach
P30	Extracting Software Product Lines: A Case Study Using Conditional Compilation	Couto, M. V. Valente, M. T. Figueiredo, E.	2011	IEEE	Modelo	Implementação de Domínio	Pesquisa de Avaliação
P31	Feature Cohesion in Software Product Lines: An Exploratory Study	Apel, S. Beyer, D.	2011	IEEE	Modelo, Ferramenta	Arquitetura de Domínio	Pesquisa de Validação
P32	Flexible Feature Binding with Aspect J-Based Idioms	Andrade, R. Rebêlo, H. Ribeiro, M. Borba, P.	2014	Scopus	Método	Implementação de Domínio	Pesquisa de Avaliação
P33	Florida: Feature LOcatIon DAshboard for Extracting and Visualizing Feature Traces	Andam, B. Burger, A. Berger, T. Chaudron, M.	2017	ACM	Método, Ferramenta	Arquitetura de Domínio	Proposta de Solução
P34	Guidelines for Using Aspects in Product Lines	Kohut, J. Vranic, V.	2010	IEEE	Modelo	Implementação de Domínio	Pesquisa de Avaliação
P35	Investigating the Variability Impact on the Recovery of Software Product Line Architectures: An Exploratory Study	Cardoso, M. L. C. Almeida, E. M. I. Chavez, C.	2017	ACM	Medida	Arquitetura de Domínio	Pesquisa de Validação
P36	Measurement of the Complexity of Variation Points in Software Product Lines	Lin, Y., Ye, H. Tang, J.	2009	IEEE	Medida	Arquitetura de Domínio	Pesquisa de Avaliação
P37	Measuring Non-Functional Properties in Software Product Line for Product Derivation	Siegmund, N. Rosenmüller, M. Kuhlemann, M. Kästner, C. Saake, G.	2008	IEEE	Método	Análise de Produto	Pesquisa de Avaliação
P38	Measuring the Impact of Traceability on the Cost of Software Product Lines Using COPLIMO	Z. Mcharfi B. El Asri I. Dehmouch A. Kriouile	2015	IEEE	Medida	Implementação de Domínio	Pesquisa de Avaliação
P39	MERCI: A Method to Evaluate Combinatorial Interaction Testing Tools for Software Product Lines	Campos, D. Lima, C. Machado, I. do C.	2018	ACM	Método	Teste de Domínio	Pesquisa de Validação
P40	Metrics for Feature-Oriented Programming	Abilio R. Vale, G. Figueiredo, E. Costa H.	2016	IEEE	Medida	Implementação de Domínio	Pesquisa de Avaliação

Tabela Apêndice A-1 - Artigos Selecionados (cont.)

ID	Título	Autores	Ano	Biblioteca	Solução	Atividade SPL	Metodologia
P41	Multi-Objective Test Prioritization in Software Product Line Testing: An Industrial Case Study	Wang, S. Buchmann, D. Ali, S. Gotlieb, A. Pradhan, D. Liaaen, M.	2014	ACM	Método	Teste de Produto	Pesquisa de Validação
P42	On the Implementation of Dynamic Software Product Lines: An Exploratory Study	Carvalho, M. da Silva, M. Gomes, G. Santos, A. Machado, I. Souza, M. de Almeida, E.	2018	Science	Medida	Implementação de Domínio	Pesquisa de Validação
P43	On the Relation Between Internal and External Feature Interactions in Feature-Oriented Product Lines: A Case Study	Kolesnikov, S. Roth, J. Apel, S.	2014	ACM	Medida	Implementação de Domínio	Pesquisa de Avaliação
P44	On the Relationship of Concern Metrics and Requirements Maintainability	Conejero, J. Figueiredo, E. Garcia, A. Hernández, J. Jurado, E.	2012	Science	Medida	Arquitetura de Domínio	Pesquisa de Validação
P45	Predicting Failure-Proneness in an Evolving Software Product Line	Krishnan, S. Strasburg, C. Lutz, R. R. Goseva-Popstojanova, K. Dorman, K. S.	2013	Science	Medida	Implementação de Produto	Pesquisa de Validação
P46	Preliminary Study on the Correlation of Objective Functions to Optimize Product-Line Architectures	Santos, M. Colanzi, T. Amaral, A. Oliveira Junior, E.	2017	ACM	Medida	Arquitetura de Domínio	Pesquisa de Validação
P47	Preprocessor-Based Variability in Open-Source and Industrial Software Systems: An Empirical Study	Hunsen, C. Zhang, B. Siegmund, J. Kästner, C. Leßenich, O. Becker, M. Apel, S.	2016	Scopus	Medida	Implementação de Domínio	Pesquisa de Avaliação
P48	Preserving Architectural Styles in the Search-Based Design of Software product Line Architectures	Mariani, T. Colanzi, T. Regina Vergilio S.	2016	Science	Método	Arquitetura de Domínio	Pesquisa de Avaliação

Tabela Apêndice A-1 - Artigos Selecionados (cont.)

ID	Título	Autores	Ano	Biblioteca	Solução	Atividade SPL	Metodologia
P49	Quantifying Structural Attributes of System Decompositions in 28 Feature-Oriented Software Product Lines: An Exploratory Study	Sobernig, S. Apel, S. Kolesnikov, S. Siegmund, N.	2016	Scopus	Medida	Implementação de Domínio	Pesquisa de Validação
P50	Setting Up Architectural SW Health Builds in a New Product Line Generation	Boss, B. Tischer, C. Krishnan, S. Nutakki, A. Gopinath, V.	2016	ACM	Modelo	Arquitetura de Domínio	Experiencia Pessoal
P51	Some Metrics for Accessing Quality of Product Line Architecture	Tao, Z. Lei, D. Jian, W. Qiaoming, Z. Chunyan, M.	2008	IEEE	Medida	Arquitetura de Domínio	Proposta de Solução
P52	Systematic Evaluation of Software Product Line Architectures	Oliveira Junior, E. Gimenes, I. Maldonado, J. Masiero, P. Barroca, L.	2013	Scopus	Método	Arquitetura de Domínio	Pesquisa de Validação
P53	TDTool: Threshold Derivation Tool	Veado, L. Vale, G. Fernandes, E. Figueiredo, E.	2016	ACM	Método, Ferramenta	Implementação de Domínio	Pesquisa de Validação
P54	The Development of a Component Commonality Metric for Mass Customization	Blecker, T. Abdelkafi, N.	2007	IEEE	Medida	Arquitetura de Domínio	Pesquisa de Avaliação
P55	Towards Indicators of Instabilities in Software Product Lines: An Empirical Evaluation of Metrics	Cafeo, B. Dantas, F. Cirilo, E. J. R. Garcia, A.	2013	IEEE	Medida	Implementação de Domínio	Pesquisa de Validação
P56	Towards System Analysis with Variability Model Metrics	Berger, T. Guo, J.	2014	Scopus	Medida, Ferramenta	Arquitetura de Domínio	Pesquisa de Validação
P57	Towards the Integration of Quality Attributes into a Software Product Line Cost Model	Nolan, A. Abrahão, S. Clements, P. McGregor, J. Cohen, S.	2011	IEEE	Modelo	Implementação de Domínio	Proposta de Solução
P58	Towards Validating Complexity-Based Metrics for Software Product Line Architectures	Marcolino, A. Oliveira Junior, E. Gimenes, I. Conte, T	2013	IEEE	Medida	Arquitetura de Domínio	Pesquisa de Validação

Tabela Apêndice A-1 - Artigos Selecionados (cont.)

ID	Título	Autores	Ano	Biblioteca	Solução	Atividade SPL	Metodologia
P59	Variability Evolution and Erosion in Industrial Product Lines: A Case Study	Bo Zhang Becker, M. Patzke T., Sierszecki K. Savolainen J	2013	ACM	Medida, Ferramenta	Implementação de Domínio	Pesquisa de Avaliação
P60	Variability in Usability Tests for Android Applications	Enriquez f. Casas S	2017	ACM	Modelo, Ferramenta	Implementação de Domínio	Pesquisa de Avaliação

APÊNDICE B - COMPILAÇÃO DOS RESULTADOS DO MSL

Tabela Apêndice B-1 - Medidas de Software Identificadas

Nome	Sigla	Característica	Aplicação	Referência	Qtde
Abstract Method	AM	Code smell	OA	P22	1
Abstractness	A	Abstração	OO	P34	1
Activated Features by Context Adaptation	AFCA	Personalização	FM	P7, P25	2
Adaptation Adjustment Modifier	AAM	Complexidade / Tamanho / Acoplamento / Coesão	OC	P38	1
Afferent Couplings	Ca	Acoplamento	OO	P34, P59, p6	3
Age of a File	Age	Code Churn	OA / OC / OO	P14, P19, P45	3
Anonymous Pointcut	AP	Code smell	OA	P22	1
Architectural Commonality	AC	Comunalidade	Meta-model	P3	1
Architecture variability	AV	Variabilidade	Meta-model	P51	1
Aspect Size	AS	Tamanho	OA	P22	1
Assessment and Assimilation factor	AA	Complexidade / Tamanho / Acoplamento / Coesão	OC	P38	1
Attribute	Att	Tamanho	OO	P30, P40	2
Average Block Depth		Complexidade	OO	P1, P19	2
Average CodeChurn		Code churn	OA / OC / OO	P9, P14, P19	3
Average Complexity		Complexidade	OA / OC / OO	P9, P19	2
Average Cyclomatic Complexity	ACC	Complexidade	OO	P1	1
Average Depth Inheritance Hierarchy	ADIT	Herança	OO	P1	1
Average FileChurn	-	Code churn	OA / OC / OO	P9	1
Average Lines of Code Added per Revision		Code churn	OA / OC / OO	P14, P19, P45	3
Average Lines of Code Deleted per Revision		Code churn	OA / OC / OO	P14, P19, P45	3
Average Lines of Code per Method		Tamanho	OO	P1	1
Average Nesting Depth		Complexidade	OA / OC	P8, P33	2
Average number of files commit- ted together to the repository		Code churn	OA / OC / OO	P14, P19, P45	3
Average Statements per Method	Avg Statements per Method	Tamanho	OA / OC / OO	P19	1
Average Weighted Methods per Class	AWMC	Complexidade	OO	P1	1
Borrowed Pointcut	BP	Code smell	OA	P22	1
Branch Statements Method	Branch Statements Method	Tamanho	OA / OC / OO	P19	1
Branching Factor Max	BF	Code Churn	FM	P25, P56	2
Class Fragmentation (ClassFrag)	ClassFrag	Acoplamento	OC	P43, P49	2

Tabela Apêndice B-1 - Medidas de Software Identificadas (cont.)

Nome	Sigla	Característica	Aplicação	Referência	Qtde
ClassDependencyIn	CDepIn	Acoplamento	OO	P6	1
ClassDependencyOut	CDepOut	Acoplamento	OO	P6	1
Classes and Interfaces		Tamanho	OA / OC / OO	P19	1
Coefficient of Connectivity-Density	CoC	Complexidade	FM	P2, P7, P29,	3
Cognitive Complexity	CogC	Complexidade	FM	P7, P13, P25, P29,	4
CombinedFeature-Reference Graph Degree	RefSWO	Complexidade	OC	P43	1
comments per line	comments per line	Tamanho	OC	P37	1
Commonality	Comm	Comunalidade	FM	P2	1
Commonality Diversity Index	CDI	Comunalidade	Meta-model	P11	1
Commonality Index	CI	Comunalidade	Meta-model	P11	1
CompClass		Complexidade	Meta-model	P52, P58	2
CompInterface		Complexidade	Meta-model	P52	1
Complexity of each component of a PLA		Complexidade	Meta-model	P26	1
Complexity of Variability	CV	Complexidade	OA	P60	1
Component Compliance	CC	Acoplamento / Coesão	Meta-model	P3	1
component reuse rate	CRR	Personalização	Meta-model	P35, P51	1
Component-level Interlacing Between Features	CIBC	Acoplamento	OA / OC / OO	P6, P20, P48	3
Composition Bloat	CB	Code smell	OA	P22	1
Compound Complexity	ComC	Complexidade	FM	P13, P25, P29	3
CompPLA		Complexidade	Meta-model	P52, P58	2
Comprehensive Metric for Commonality	CMC	Comunalidade	Meta-model	P11	1
CompVarComponent		Complexidade	Meta-model	P52	1
CompVariabilityClass		Complexidade	Meta-model	P52	1
CompVarPointClass		Complexidade	Meta-model	P52	1
Concentration	Conc	Abstração	OC	P28, P40, P49	3
Concern Degree of Tangling		Entrelaçamento	OA	P44	1
Concern Diffusion over Components	CDC	Espalhamento	OA / OC	P5, P16, P17, P20, P27, P42	6
Concern Diffusion over Lines of Code	CDLOC	Entrelaçamento	OA / OC	P5, P20, P27	3
Concern Diffusion over Operations	CDO	Espalhamento	OA / OC	P5, P20, P27, P42	4
Concern Interlacing		Acoplamento	OA	P44	1
Context Features	CF	Personalização	FM	P7, P25	2
Context Features in Constraints	CFC	Personalização	FM	P7, P25	2
Coupling among classes ofa PLA	ACLASS	Acoplamento	Meta-model	P46	1
Coupling Between Components	CBC	Acoplamento	OA / OO	P10, P20, P22, P42	4
Coupling Between Features	CBF	Acoplamento	OC	P43	1
Coupling between Modules	CBM	Acoplamento	OA / OO	P34	1

Tabela Apêndice B-1 - Medidas de Software Identificadas (cont.)

Nome	Sigla	Característica	Aplicação	Referência	Qtde
Coupling Between Objects	CBO	Acoplamento	OO	P4, P23, P43, P53, P55	5
Coupling between the PLA components	ACOMP	Acoplamento	Meta-model	P46	1
Coupling Between Units	CBU	Acoplamento	OC	P49	1
Coupling of Field Access	CFA	Acoplamento	OA / OO	P34	1
Coupling of Method Call	CMC	Acoplamento	OA / OO	P34	1
Coverage of Variability	CV	Variabilidade	Meta-model	P3	1
Cross-Tree Constraints	CTC	Complexidade	FM	P13, P18, P29	3
Cross-tree constraints Variables	CTCV	Complexidade	FM	P29	1
Cross-Tree-Constraints Ratio	CTCR	Tamanho	FM	P2, P56	2
Crosscutting Degree of an Aspect	CDA	Acoplamento	OA	P34	1
Cumulative Applicability	CA	Comunalidade	Meta-model	P3	1
Cumulative Halstead Effort	HEFF	Tamanho	OO	P1	1
Cyclomatic Complexity	CyC	Complexidade	OA / OC / OO	P2, P14, P19, P26, P30, P41, P54	7
Deactivated Features by Context Adaptation	DFCA	Personalização	FM	P7, P25	2
Dedication	Dedi	Abstração	OC	P28, P40	2
Degree of Commonality Index	DCI	Comunalidade	Meta-model	P11	1
Degree of Documentation	DOCU	Tamanho	OC	P38	1
Degree of focus of scenarios	DoF	Espalhamento / Entrelaçamento	OA	P28	1
Degree of Scattering across Components	DOSC	Espalhamento	OA	P15, P16, P17, P32	4
Degree of Scattering across Operations	DOSO	Espalhamento	OA	P16, P32	2
Degree of Scattering of Features	DoS	Espalhamento	OA	P28	1
Degree of Tangling of Scenarios	DoT	Entrelaçamento	OA	P28	1
Degree of Tangling within Components	DOTC	Entrelaçamento	OA	P15, P16, P17, P32	4
Degree of Tangling within Operations	DOTO	Entrelaçamento	OA	P16, P32	2
Density of the graph	Rden	Complexidade	FM	P7, P13, P18, P29, P56	5
Dependency of Packages	DepPack	Acoplamento	Meta-model	P6	1
DependencyIn	DepIn	Acoplamento	OC / OO	P6	1
DependencyOut	DepOut	Acoplamento	OC / OO	P6	1
Depth of Inheritance Tree	DIT	Herança	OO	P34, P42, P55	3
Depth of Tree	DT	Complexidade	FM	P13	1
Design Structure Matrix	DSM	Comunalidade	Meta-model	P12	1
Development for Reuse factor	RUSE	Tamanho / Acoplamento / Coesão / Mensagem	OC	P38	1
Disparity	Disp	Abstração	OC	P40	1
Distance Between Features	Dist	Complexidade	OC	P40	1
Distance-based External-ratio Feature Dependency	EFDW	Coesão	OC	P31	1

Tabela Apêndice B-1 - Medidas de Software Identificadas (cont.)

Nome	Sigla	Característica	Aplicação	Referência	Qtde
Distance-based Internal-ratio Feature Dependency	IFDW	Coesão	OC	P31	1
Duplicate Pointcut	DP	Code smell	OA	P22	1
Dynamicity measures of the feature model	DIFM	Personalização	FM	P7	1
Effectiveness of Tailoring	ET	Personalização	Meta-model	P3	1
Efferent Couplings	Ce	Acoplamento	OO	P35, P54	2
ExtensClass		Acoplamento / Herança / Polimorfismo	Meta-model	P52	1
Extensibility of each component of a PLA		Acoplamento / Herança / Polimorfismo	Meta-model	P26	1
ExtensInterface		Acoplamento / Herança / Polimorfismo	Meta-model	P52	1
ExtensPLA		Acoplamento / Herança / Polimorfismo	Meta-model	P52	1
ExtensVarComponent		Acoplamento / Herança / Polimorfismo	Meta-model	P52	1
ExtensVariabilityClass		Acoplamento / Herança / Polimorfismo	Meta-model	P52	1
ExtensVarPointClass		Acoplamento / Herança / Polimorfismo	Meta-model	P52	1
Exterior information flow complexity	EIFC	Complexidade	Meta-model	P51	1
External-ratio Feature Dependency	EFD	Coesão	OC	P31	1
External-ratio Unit Dependency	EUD	Coesão	OC	P49	1
Fan-in		Acoplamento	OA	P59	1
Fan-out		Acoplamento	OA	P59	1
Fault Detection Capability	FDC	CObertura	FM	P21, P41	2
Feature Diffusion over Architectural Components	CDAC	Espalhamento	OC / OO	P6, P48	2
Feature Diffusion over Architectural Interfaces	CDAI	Espalhamento	OC / OO	P6, P48	2
Feature Diffusion over Architectural Operations	CDAO	Espalhamento	OC / OO	P6, P48	2
Feature Envy	FEX	Acoplamento	OA	P22	1
Feature EXtendibility	FEX	Abstração / Acoplamento / Herança	FM	P13, P25, P29	3
Feature fragmentation	FeatureFrag	Acoplamento	OC	P43	1
Feature pairwise coverage	FPC	Cobertura	FM	P21, P41	2
Feature-reference and structure graph degree	RefS	Acoplamento	OC	P43	1
Feature-reference graph degree	Ref	Acoplamento	OC	P40, P43	2
Feature-Reference-and-Optionality Graph Degree	RefO	Complexidade	OC	P43	1
FileChurn	FileChurn	Code churn	OA / OC / OO	P9	1
Flexibility Index of the Feature Model	FIFM	Complexidade	FM	P7	1

Tabela Apêndice B-1 - Medidas de Software Identificadas (cont.)

Nome	Sigla	Característica	Aplicação	Referência	Qtde
Flexibility of Configuration	FoC	Complexidade	FM	P7, P13, P18, P25, P29	5
Forced Join Point	FJP	Acoplamento	OA	P22	1
Fraction of annotated lines of code	PLoF	Variabilidade	OA	P47	1
Functional Commonality	FC	Comunalidade	OA / OC / OO	P60	1
Functional Coverage	FC	Comunalidade	Meta-model	P3	1
Functional Similarity Index	FSI	Comunalidade	Meta-model	P11	1
Generational Variety Index	GVI	Comunalidade	Meta-model	P11	1
Global scope	GoS	Acoplamento	OO	P55	1
God Aspect		Code smell	OA	P22	1
God Class		Code smell	OC	P4, P22	2
God Method		Code Smell	OC	P24	1
God Pointcut		Code smell	OA	P22	1
Heterogeneous	HEM	Variabilidade	FM	P56	1
Homogeneous	HOM	Comunalidade	FM	P56	1
Hybrid Feature Coupling	HFC	Acoplamento	OC	P40	1
Idle Pointcut	IdP	Code smell	OA	P22	1
Improvements	Improvements	Code churn	OA / OC / OO	P9	1
Instability	I	Acoplamento	OO	P34	1
Instability Metric	IM	Acoplamento	OA / OO	P42	1
Interface-level Interlacing Between Features	IIBC	Acoplamento	OC / OO	P6, P48	2
Interior information flow complexity	IIFC	Complexidade	Meta-model	P51	1
Internal Ratio Unit Dependency	IUD	Acoplamento	OC	P49	1
Internal-ratio Feature Dependency	IFD	Coesão	OC	P31, P43	2
Introduction	AMI	Code smell	OA	P22	1
Lack of Cohesion in Methods	LCOM	Coesão	OO	P55	1
Lack of cohesion in operations	LCOO	Coesão	OA	P10, P20, P42	3
Lack of Cohesion over Operations	LCO	Coesão	OA	P34	1
lack of feature-based cohesion	LCC	Coesão	Meta-model	P46	1
Lack of Feature-based Cohesion(LCC)	LCC	Coesão	OA / OC / OO	P6, P48	2
Lazy Aspect	LA	Code smell	OA	P22	1
Lazy Class		Code smell	OC	P4	1
Lines of Code	LOC	Tamanho	OA / OC / OO	P4, P8, P9, P10, P19, P20, P22, P23, P27, P30, P34, P37, P40, P43, P47, P53	16
Lines of Concern Code	LOCC	Tamanho	OA / OC	P5, P27	2
Lines of Feature Code	LOF	Tamanho	OA / OC	P8, P30, P33, P47	4
Lines with Comments	Lines with Comments	Tamanho	OA / OC / OO	P19	1

Tabela Apêndice B-1 - Medidas de Software Identificadas (cont.)

Nome	Sigla	Característica	Aplicação	Referência	Qtde
Local scope	Los	Acoplamento	OO	P55	1
Maintanability Index	MI	Halstead	OO	P1, P57	2
Maturity		Personalização	Meta-model	P57	1
Maximum Block Depth		Complexidade	OA / OC / OO	P19	1
Maximum CodeChurn		Code churn	OA / OC / OO	P14, P19, P45	3
Maximum Complexity		Complexidade	OA / OC / OO	P9, P19	2
Maximum Depth of Tree		Complexidade	FM	P7, P25, P29,	3
Maximum number of files com- mitted together to the reposi- tory		Code churn	OA / OC / OO	P14, P19, P45	3
Maximum number of lines of code added for all revisions		Code churn	OA / OC / OO	P14, P19, P45	3
Maximum number of lines of code deleted for all revisions		Code churn	OA / OC / OO	P14, P19, P45	3
Maximum Textual Feature Coupling	TFCMax	Acoplamento	OC	P40	1
Mean Depth of Tree	DTMean	Complexidade	FM	P25, P29	2
Median Depth of Tree	DTMedian	Complexidade	FM	P25, P29	2
Method Call Statements	Method Call Statements	Tamanho	OA / OC / OO	P19	1
Methods per Class Ave	Methods per Class Ave	Tamanho	OA / OC / OO	P19	1
Multiple Cyclic Dependent Features	MCDF	Acoplamento	FM	P13, P25, P29	3
Multiple Hotspot Features	MHoF	Acoplamento	FM	P13, P25, P29	3
Nesting Depth	ND	Complexidade	OO	P47	1
Nesting Level	NEST	Complexidade	OA	P59	1
New Features	NewFeatures	Tamanho / Complexidade	OA / OC / OO	P9	1
NMutex		Tamanho	FM	P56	1
Non-Functional Commonality	NFC	Comunalidade	Meta-model	P3	1
Non-functional Coverage	NC	Comunalidade	Meta-model	P3	1
Normalized Average Radius	NAR	Coesão	OC	P31	1
Normalized Maximum Radius	NMR	Coesão	OC	P31, P40	2
Number of Access Attributes	NAA	Tamanho	OA	P22	1
Number of Activated Features	NAF	Tamanho	FM	P25	1
Number of Affected Classes	NAC	Tamanho	OC	P33	1
Number of Alternative Features	NA	Variabilidade	FM	P13	1
Number of Aspects shared Join Point	NAsJP	Tamanho	OA	P22	1
Number of Attributes	NOA	Tamanho	OA	P10	1
Number of Children	NOC	Herança / Abstração	OO	P34	1
Number of Children per Feature	BFMax	Complexidade	FM	P29	1
Number of Classes	NCLASS	Tamanho	Meta-model	P30, P43	2
Number of Classes implementing the Optional Features	ClassOptional	Tamanho	Meta-model	P35	1
Number of Components	NOC	Tamanho	OA / OC / OO	P10, P20, P40, P55	4

Tabela Apêndice B-1 - Medidas de Software Identificadas (cont.)

Nome	Sigla	Característica	Aplicação	Referência	Qtde
Number of compositions items add / del		Tamanho	OA	P28	1
Number of Constant Refinements	NCR	Tamanho	OC	P4, P23, P24, P40, P53	5
Number of Constants	NOCt	Tamanho	OC	P40	1
Number of Constants Refinemets	NRC	Complexidade	OC	P40	1
Number of context	NC	Tamanho	FM	P25	1
Number of Context Constraints	NCC	Tamanho	FM	P25	1
Number of Deactivated Features	NDF	Tamanho	FM	P25	1
Number of distinct authors that made revisions to the fil	AUT	Tamanho	OA / OC / OO	P14, P19, P45	3
Number of distinct references inside the component that depend upon classes external components	D5	Acoplamento	OA / OC / OO	P1	1
Number of distinct references inside the component that depend upon classes within environment	D4	Acoplamento	OA / OC / OO	P1	1
Number of distinct references outside the plataform that depend upon classes within the plataform	D3	Acoplamento	OA / OC / OO	P1	1
Number of Features	NOF	Tamanho	OC	P24, P33	2
Number of Features Constants	NOFC	Tamanho	OA	P8, P56	2
Number of Features FM	NF	Complexidade / Tamanho	FM / Meta-modelo	P7, P13, P18, P25, P29, P39, P40, P56	8
Number of Features with Code	NFC	Tamanho	OC	P40	1
Number of File Annotations	NoFiA	Tamanho	OA	P33	1
Number of Files	NFiles	Tamanho	OA / OC / OO	P9	1
Number of Files Modified	NoFi	Tamanho	OA	P33	1
Number of Folder Annotations	NoFoA	Tamanho	OA / OO	P34	1
Number of grouping features	NGF	Complexidade	FM	P29, P56	2
Number of Leaf Features	NLeaf	Tamanho	FM	P7, P13, P18, P25, P29, P56	6
Number of Mandatory Features	NM	Tamanho	FM	P7, P25, P29	3
Number of Mandatory releationship Ror	Rand	Tamanho	FM	P13	1
Number of Method Refinements	NMR	Complexidade	OC	P24,	1
Number of Methods	NOM	Tamanho	OO	P1, P39	2
Number of operations	NOO	Tamanho	OA	P10, P20	2
Number of Operations by Interface	NumOps	Tamanho	OC / OO	P6	1
Number of Optional Features	NO	Variabilidade	FM / Meta-modelo	P13, P25, P29, P39	4
Number of Or features	NOr	Tamanho	FM	P56	1
Number of OR relationship	Ror	Tamanho	FM	P13, P25	2
Number of Packages	NOP	Tamanho	OA	P30	1
Number of products	NP	Tamanho	Meta-model	P35	1

Tabela Apêndice B-1 - Medidas de Software Identificadas (cont.)

Nome	Sigla	Característica	Aplicação	Referência	Qtde
Number of Refined Methods	NRM	Complexidade	OC	P40	1
Number of Refinemets	NoR	Tamanho	OC	P40	1
Number of relationships among nodes including constrains	R	Tamanho	FM	P13	1
Number of revisions made to a fil	Revisions	Tamanho	OA / OC / OO	P14, P19, P45	3
Number of Statements	NOS	Tamanho	OA / OC / OO	P1	1
Number of steps add / del between 2 releases		Tamanho	OA	P28	1
Number of times a file has been refactored	Refactoring	Tamanho	OA / OC / OO	P14, P19, P45	3
Number of times a file was in- volved in bug-fixin	BugFixes	Tamanho	OA / OC / OO	P14, P19, P45	3
Number of Top Features	NTop	Complexidade / Tamanho	FM	P13, P18, P25, P29, P56	5
Number of Types	NOT	Tamanho	OA / OO	P34	1
Number of Valid Configurations	NVC	Variabilidade / Complexidade	FM	P7, P13, P18, P25, P29	5
number of variable components found on PLA	PLTotalVariability	Tamanho	Meta-model	P35	1
Number of Variable Features	NVF	Variabilidade	FM	P13, P25, P29	3
Number of variation points	NoVP	Complexidade	OA	P59	1
Number of XOR or OR relationship	Rgr	Tamanho	FM	P13	1
Number of XOR relationship	Rcase	Tamanho	FM	P13	1
number ofclasses implementing the mandatory features	ClassMandatory	Tamanho	Meta-model	P35	1
Number OR Groups	NGOr	Tamanho	FM	P25, P29	2
Number XOR Groups	NGXOr	Tamanho	FM	P25, P29	2
Numberof configuration items add / del		tamanho	OA	P28	1
Numer of Xor features	NXor	Tamanho	FM	P56	1
Operation-level Overlapping Between Features	OOBC	Acoplamento	OC	P6, P48	2
Pairs of Cloned Code	PCC	Clone	OA	P15, P16, P17, P32	4
Percent Commonality	%C	Comunalidade	Meta-model	P11	1
PLA relational cohesion	COE	Coesão	Meta-model	P46	1
PLA Tamanho	TAM	Tamanho	Meta-model	P46	1
PLA-IFG cyclomatic complexity	PCC	Complexidade	Meta-model	P51	1
PLA-IFG information flow complexity	PIFC	Complexidade	Meta-model	P51	1
PLA-IFG total complexity	PTC	Complexidade	Meta-model	P51	1
Product line commonality index	PCI	Comunalidade	Meta-model	P12	1
Produto entre age e Loc_Add		Code churn	OA / OC / OO	P14, P19, P45	3
Ratio between the number of composition composition items and number of matched join points		Comunalidade	OA	P28	1
Ratio of Assets to Features	RAF	Tamanho / Complexidade	OA / OC / OO	P60	1
Ratio of connectivity of this graph	RCon	Complexidade	FM	P29, P56	2
Ratio of Variability	RoV	Variabilidade / Complexidade	FM	P7, P13, P18, P25, P29,	5

Tabela Apêndice B-1 - Medidas de Software Identificadas (cont.)

Nome	Sigla	Característica	Aplicação	Referência	Qtde
RConstr		Tamanho	FM	P56	1
RDefault		Tamanho	FM	P56	1
RDefaultExpr		Tamanho	FM	P56	1
RDefaultLit		Tamanho	FM	P56	1
RDerived		Tamanho	FM	P56	1
RDerivedExpr		Tamanho	FM	P56	1
RDerivedLit		Tamanho	FM	P56	1
Redundant Pointcut	RP	Code smell	OA	P22	1
Relational Cohesion	H	Coesão	OC / OO	P6	1
Relative Cost of Reuse	RCR	Tamanho / Acoplamento / Coesão / Mensagem	OC	P38	1
Relative Cost of Writing for Reuse	RCWR	Tamanho / Acoplamento / Coesão / Mensagem	OC	P38	1
Requirements Reliability	RELY	Complexidade / Tamanho / Acoplamento / Coesão	OC	P38	1
Response for a Class	RFC	Complexidade	OO	P34, P42, P55	3
Reusability	RE	Tamanho / Acoplamento / Coesão / Mensagem	Meta-model	P3	1
Reuse Benefit Rate	RBR	Tamanho	Meta-model	P51	1
Rigid Nohotspot Features	RNoF	Variabilidade	FM	P13, P25, P29	3
RPurelyBoolConstr		Tamanho	FM	P56	1
RVisibility		Tamanho	FM	P56	1
Scattering Degree	SD	Espalhamento	OA / OC	P8, P30, P33, P40, P44, P46, P47	7
Set of Primitive Pointcuts	SPP	Tamanho	OA	P22	1
Shotgun Surgery		Code smell	OC	P24	1
similarity between PLA components.	SSC	Similaridade	Meta-model	P51	1
Single Cyclic Dependent Features	SCDF	Acoplamento	FM	P13, P25, P29	3
Single Hotspot Features	SHoF	Variabilidade	FM	P13, P25, P29	3
Size Index of the Feature Model	SIFM	Tamanho	OA	P7	1
Software Understanding	SU	Complexidade / Tamanho / Acoplamento / Coesão	OC	P38	1
Source Lines of Code	SLOC	Tamanho	OA	P15, P16, P17, P32, P49	5
STability Index of the Feature Model	STIFM	Variabilidade / Complexidade	FM	P7	1
Statements	Statements	Tamanho	OA / OC / OO	P19	1
Strong Coupling of Variability	SCC	Acoplamento	Meta-model	P51	1
Structural Feature Coupling	SFC	Acoplamento	OC	P40	1

Tabela Apêndice B-1 - Medidas de Software Identificadas (cont.)

Nome	Sigla	Característica	Aplicação	Referência	Qtde
Structure Variability Coefficient	SCV	Variabilidade	Meta-model	P51	1
Sum of added/deleted lines of code over all revisions	CodeChurn	Code churn	OA / OC / OO	P9, P14, P19	3
Sum over all revisions of the number of lines of code added to the fil		Code churn	OA / OC / OO	P14, P19, P45	3
Sum over all revisions of the number of lines of code deleted from the fil		Code churn	OA / OC / OO	P14, P19, P45	3
TaiLorability	TL	Personalização	Meta-model	P3	1
Tailorability of Closed variability	TC	Personalização	Meta-model	P3	1
Tailorability of Open variability	TO	Personalização	Meta-model	P3	1
Tangling Degree	TD	Entrelaçamento	OA / OC	P8, P30, P33, P40, P44, P46, P47	7
Textual Feature Coupling	TFC	Acoplamento	OC	P40	1
Total Commonality Index	TCI	Comunalidade	Meta-model	P54	1
Total Constant Commonality Index	TCCI	Comunalidade	Meta-model	P11	1
Total Cyclomatic Complexity	TCC	Complexidade	OA / OC / OO	P1	1
Total Number of Constants Number	TNC	Tamanho	OC	P24	1
Total Number of Method Refinement	TNMR	Tamanho	OC	P24	1
Total Number of Refined Constants Total	TNRC	Tamanho	OC	P24	1
Total Number of Refined Methods	TNRM	Tamanho	OC	P24	1
Total Number of Refinements	TNR	Tamanho	OC	P24	1
Variability Coverage	VC	Variabilidade	FM	P2	1
Variability Point	VP	Variabilidade	OA	P28	1
VariationRank	VR	Variabilidade	FM	P36	1
Various Concerns	VC	Code smell	OA	P22	1
Vocabulary Tamanho	VS	Tamanho	OA	P15, P16, P17, P32	4
vVariability Between PLA Components	SVC	Variabilidade	Meta-model	P35	1
Weak Coupling Coefficient	WCC	Acoplamento	Meta-model	P51	1
Weighted Feature-reference Graph Degree	RefW	Acoplamento	OC	P43	1
Weighted Operations in Module	WOM	Complexidade	OA / OO	P34	1
Wheighted Methods per Class	WMC	Complexidade	OO	P4, P23, P53, P55	4
Wighted Operation Count	WOC	Tamanho	OA / OO	P42	1
XOr Feature Ratio	RXor	Tamanho	FM	P25, P29	2

Tabela Apêndice B-2 - Sistemas de Software Acadêmicos Utilizados

Ocorrências	Sistemas de Software
11	BerkeleyDB
9	MobileMedia
6	GPL PokerSPL TankWar
5	AGM ArgoUML-SPL BankAccount EPL
4	DesktopSearcher ExamDB Notepad Prop4J UnionFind
3	AHEAD-Bali AHEAD-guidsl AHEAD-Java Arcade Game BET Devolution Digraph Elevator EmailSystem Paycard Sudoku GameOfLite Vistex
2	101Companies BALI2JAK BALI2JAVACC BALI2LAYER BALICOMPOSER BCJAK2JAVA Eclipse-C/C++ Eclipse-Classic GPLscratch GUIDSL HealthWatcher HelloWorld IntegerSet IntList JAK2JAVA JRENAME MinePump MIXIN MM MMATRIX

Ocorrências	Sistemas de Software
2	NumberConsecutiveContractRef NumberContractOverrinding NumberExplicitContractRef PREVAYLER SmartHome Stack StringMatcher UNMIXIN VIOLET WebStore Zip Me
1	AHEAD AJStats Bali BAN Bicycle clamav DELL Dia Documentation generation DPL E-Commerce Electronic shopping FAME-DBMS GOL HIS Inventory JAMPACK Java Email Server JHotDraw LinkedList MobileGame MobileGuide Model transformation MOM PKJab Raroscope Search engine Sienna SmartHomeRiSE Tetris Text editor Thread VideoPlayer VOD Web portal

Tabela Apêndice B-3 - Sistemas de Software de Código Aberto Utilizados

Ocorrências	Sistemas de Software
4	Freemind
3	Eclipse-Java Eclipse-JavaEE
2	CHEROKEE Eclipse-C/C++ Eclipse-Classic FREEBSD GIMP GNUMERIC GNUPLOT

Ocorrências	Sistemas de Software
2	LIBXML2 LINUX OPENVPN PARROT POSTGRESQL SENDMAIL SQLITE SUBVERSION VIM XFIG XTERM

Tabela Apêndice B-3 - Sistemas de Software de Código Aberto Utilizados (Cont.)

Ocorrências	Sistemas de Software
1	BUSYBOX emacs gcc ghostscript glibc irssi lighttpd lynx minix mplayer mpsolve

Ocorrências	Sistemas de Software
1	openldap opensolaris php pidgin privoxy python QEMU sylpheed tcl xine-lib xorg-server

APÊNDICE C - DADOS BRUTOS DAS MEDIDAS DE SOFTWARE

Tabela Apêndice C-1 - Dados Brutos das Medidas - Tecnologia OC

Características	DepIn	DepOut	SFC	EFD	IFD	LCC	LOC	NOA	NOO
Negrito	0	2	0,06	0	0	3	20	0	3
Highlight	0	1	0,08	0	0	2	24	0	4
Enorme	0	1	0,02	0	0	2	26	0	1
OpenDyslexic	0	1	0,02	0	0	2	10	0	1
Tamanho	13	1	0,23	0,75	0,19	2	105	9	13
WebHelp	12	0	0,14	1	0,25	1	117	13	11
Texto	5	1	0,2	0,67	0,22	2	70	6	9
Caracteres	0	2	0,07	0	0	3	20	0	3
Alinhamento	0	2	0,06	0	0	3	20	0	3
Leitor	0	1	0,07	0,33	0,11	2	36	0	4
Fonte	0	3	0,11	0	0	4	45	0	5
OpenSans	0	1	0,02	0	0	2	10	0	1
Overlay	0	1	0,13	0	0	2	30	0	3
Linhas	0	2	0,06	0	0	3	20	0	3
Sublinhado	0	2	0,06	0	0	3	20	0	3
Background	0	2	0,09	0	0	3	24	0	3
Grande	0	1	0,02	0	0	2	26	0	1
Cor	4	1	0,21	0,83	0,2	2	102	7	14
Paragrafos	0	2	0,07	0	0	3	21	0	3
Regua	0	1	0,11	0	0	2	33	0	3
Pequeno	0	1	0,02	0	0	2	26	0	1
ComicSans	0	1	0,02	0	0	2	10	0	1
Georgia	0	1	0,02	0	0	2	10	0	1
Italico	0	2	0,06	0	0	3	20	0	3
Medio	0	1	0,06	0	0	2	26	0	1
Total	**34**	**34**	**2,01**	**3,58**	**0,97**	**59**	**871**	**35**	**98**

Tabela Apêndice C-2 - Dados Brutos das Medidas - Tecnologia OA

Características	DepIn	DepOut	SFC	EFD	IFD	LCC	LOC	NOA	NOO
Negrito	0	2	0,04	0	0	3	7	1	2
Highlight	0	2	0,05	0	0	3	8	2	3
Enorme	5	1	0	0	0	2	6	0	1
OpenDyslexic	1	1	0	0	0	2	6	0	1
Tamanho	5	1	0,18	0,5	0,22	2	98	6	13
WebHelp	24	0	0,12	1	0,25	1	107	11	10
Texto	5	1	0,16	0,5	0,22	2	72	5	10
Caracteres	0	6	0,07	0	0	7	8	2	3
Alinhamento	0	2	0,04	0	0	3	7	1	2
Leitor	0	2	0,03	0,33	0,25	3	22	1	3
Fonte	0	11	0,1	0	0	12	10	4	6
OpenSans	1	1	0	0	0	2	6	0	1
Overlay	0	2	0,1	0	0	3	8	2	3
Linhas	0	6	0,07	0	0	7	8	2	3
Sublinhado	0	2	0,04	0	0	3	4	1	2
Background	0	2	0,08	0	0	3	8	2	3
Grande	5	1	0	0	0	2	6	0	1
Cor	4	1	0,14	0,67	0,25	2	96	5	14
Paragrafos	0	6	0,07	0	0	7	8	2	3
Regua	0	6	0,1	0	0	7	8	2	3
Pequeno	5	1	0	0	0	2	6	0	1
ComicSans	1	1	0	0	0	2	6	0	1
Georgia	1	1	0	0	0	2	6	0	1
Italico	0	2	0,04	0	0	3	7	1	2
Medio	5	1	0	0	0	2	6	0	1
Total	**62**	**62**	**1,43**	**3**	**1,19**	**87**	**534**	**50**	**93**

Tabela Apêndice C-3 - Dados Brutos das Medidas - Tecnologia MCA

Características	DepIn	DepOut	SFC	EFD	IFD	LCC	LOC	NOA	NOO
Negrito	0	1	0,06	0	0	2	12	1	3
Highlight	0	1	0,07	0	0	2	12	1	3
Enorme	0	1	0,02	0	0	2	26	0	1
OpenDyslexic	0	1	0,02	0	0	2	13	0	1
Tamanho	13	1	0,22	0,75	0,33	2	104	8	14
WebHelp	3	0	0,14	1	0,25	1	117	13	11
Texto	5	1	0,19	0,67	0,5	2	66	5	9
Caracteres	0	1	0,06	0	0	2	11	1	3
Alinhamento	0	1	0,05	0	0	2	4	1	2
Leitor	0	1	0,03	0,5	0,25	2	19	1	3
Fonte	0	2	0,1	0	0	3	17	1	4
OpenSans	0	1	0,02	0	0	2	13	0	1
Overlay	0	1	0,12	0	0	2	12	1	3
Linhas	0	1	0,06	0	0	2	14	1	3
Sublinhado	0	1	0,06	0	0	2	12	1	3
Background	0	1	0,07	0	0	2	15	1	3
Grande	0	1	0,02	0	0	2	26	0	1
Cor	4	1	0,19	0,83	0,31	2	98	6	14
Paragrafos	0	1	0,06	0	0	2	11	1	3
Regua	0	1	0,1	0	0	2	15	1	3
Pequeno	0	1	0,02	0	0	2	29	0	1
ComicSans	0	1	0,02	0	0	2	13	0	1
Georgia	0	1	0,02	0	0	2	13	0	1
Italico	0	1	0,06	0	0	2	12	1	3
Medio	0	1	0,02	0	0	2	26	0	1
Total	**25**	**25**	**1,8**	**3,75**	**1,64**	**50**	**710**	**45**	**95**

APÊNDICE D - DADOS NORMALIZADOS

Tabela Apêndice D-1 - Dados Normalizados das Medidas - Tecnologia OC

Características	DepIn	DepOut	SFC	EFD	IFD	LCC
Negrito	-0,38	1,00	-0,30	-0,46	-0,47	1,00
Highlight	-0,38	-0,56	0,02	-0,46	-0,47	-0,56
Enorme	-0,38	-0,56	-0,95	-0,46	-0,47	-0,56
OpenDyslexic	-0,38	-0,56	-0,95	-0,46	-0,47	-0,56
Tamanho	3,25	-0,56	2,43	1,96	1,84	-0,56
WebHelp	2,97	2,13	0,98	2,77	2,57	-2,13
Texto	1,02	-0,56	1,94	1,70	2,20	-0,56
Caracteres	-0,38	1,00	-0,14	-0,46	-0,47	1,00
Alinhamento	-0,38	1,00	-0,30	-0,46	-0,47	1,00
Leitor	-0,38	-0,56	-0,14	0,60	0,87	-0,56
Fonte	-0,38	2,57	0,50	-0,46	-0,47	2,57
OpenSans	-0,38	-0,56	-0,95	-0,46	-0,47	-0,56
Overlay	-0,38	-0,56	0,82	-0,46	-0,47	-0,56
Linhas	-0,38	1,00	-0,30	-0,46	-0,47	1,00
Sublinhado	-0,38	1,00	-0,30	-0,46	-0,47	1,00
Background	-0,38	1,00	0,18	-0,46	-0,47	1,00
Grande	-0,38	-0,56	-0,95	-0,46	-0,47	-0,56
Cor	0,74	-0,56	2,10	2,22	1,96	-0,56
Paragrafos	-0,38	1,00	-0,14	-0,46	-0,47	1,00
Regua	-0,38	-0,56	0,50	-0,46	-0,47	-0,56
Pequeno	-0,38	-0,56	-0,95	-0,46	-0,47	-0,56
ComicSans	-0,38	-0,56	-0,95	-0,46	-0,47	-0,56
Georgia	-0,38	-0,56	-0,95	-0,46	-0,47	-0,56
Italico	-0,38	1,00	-0,30	-0,46	-0,47	1,00
Medio	-0,38	-0,56	-0,95	-0,46	-0,47	-0,56

Tabela Apêndice D-2 - Dados Normalizados das Medidas - Tecnologia OA

Características	DepIn	DepOut	SFC	EFD	IFD	LCC
Negrito	-0,50	-0,19	-0,32	-0,45	-0,49	-0,19
Highlight	-0,50	-0,19	-0,13	-0,45	-0,49	-0,19
Enorme	0,51	-0,58	-1,05	-0,45	-0,49	-0,58
OpenDyslexic	-0,30	-0,58	-1,05	-0,45	-0,49	-0,58
Tamanho	0,51	-0,58	2,25	1,43	1,77	-0,58
WebHelp	4,33	-0,98	1,15	3,31	2,07	-0,98
Texto	0,51	-0,58	1,88	1,43	1,77	-0,58
Caracteres	-0,50	1,39	0,24	-0,45	-0,49	1,38
Alinhamento	-0,50	-0,19	-0,32	-0,45	-0,49	-0,19
Leitor	-0,50	-0,19	-0,50	0,79	2,07	-0,19
Fonte	-0,50	3,36	0,79	-0,45	-0,49	3,36
OpenSans	-0,30	-0,58	-1,05	-0,45	-0,49	-0,58
Overlay	-0,50	-0,19	0,79	-0,45	-0,49	-0,19
Linhas	-0,50	1,39	0,24	-0,45	-0,49	1,38
Sublinhado	-0,50	-0,19	-0,32	-0,45	-0,49	-0,19
Background	-0,50	-0,19	0,42	-0,45	-0,49	-0,19
Grande	0,51	-0,58	-1,05	-0,45	-0,49	-0,58
Cor	0,31	-0,58	1,52	2,07	2,07	-0,58
Paragrafos	-0,50	1,39	0,24	-0,45	-0,49	1,00
Regua	-0,50	1,39	0,79	-0,45	-0,49	-0,56
Pequeno	0,51	-0,58	-1,05	-0,45	-0,49	-0,58
ComicSans	-0,30	-0,58	-1,05	-0,45	-0,49	-0,58
Georgia	-0,30	-0,58	-1,05	-0,45	-0,49	-0,58
Italico	-0,50	-0,19	-0,32	-0,45	-0,49	-0,19
Medio	0,51	-0,58	-1,05	-0,45	-0,49	-0,58

Tabela Apêndice D-3 - Dados Normalizados das Medidas - Tecnologia MCA

Características	DepIn	DepOut	SFC	EFD	IFD	LCC
Negrito	-0,35	0,00	-0,20	-0,48	-0,47	0,00
Highlight	-0,35	0,00	-0,03	-0,48	-0,47	0,00
Enorme	-0,35	0,00	-0,88	-0,48	-0,47	0,00
OpenDyslexic	-0,35	0,00	-0,88	-0,48	-0,47	0,00
Tamanho	4,20	0,00	2,51	1,90	1,88	0,00
WebHelp	0,70	-3,46	1,15	2,69	1,31	3,46
Texto	1,40	0,00	2,00	1,65	3,09	0,00
Caracteres	-0,35	0,00	-0,20	-0,48	-0,47	0,00
Alinhamento	-0,35	0,00	-0,37	-0,48	-0,47	0,00
Leitor	-0,35	0,00	-0,71	1,11	1,31	0,00
Fonte	-0,35	3,46	0,48	-0,48	-0,47	3,46
OpenSans	-0,35	0,00	-0,88	-0,48	-0,47	0,00
Overlay	-0,35	0,00	0,82	-0,48	-0,47	0,00
Linhas	-0,35	0,00	-0,20	-0,48	-0,47	0,00
Sublinhado	-0,35	0,00	-0,20	-0,48	-0,47	0,00
Background	-0,35	0,00	-0,03	-0,48	-0,47	0,00
Grande	-0,35	0,00	-0,88	-0,48	-0,47	0,00
Cor	1,00	0,00	2,00	2,16	1,74	0,00
Paragrafos	-0,35	0,00	-0,20	-0,48	-0,47	0,00
Regua	-0,35	0,00	0,48	-0,48	-0,47	0,00
Pequeno	-0,35	0,00	-0,88	-0,48	-0,47	0,00
ComicSans	-0,35	0,00	-0,88	-0,48	-0,47	0,00
Georgia	-0,35	0,00	-0,88	-0,48	-0,47	0,00
Italico	-0,35	0,00	-0,20	-0,48	-0,47	0,00
Medio	-0,35	0,00	-0,88	-0,48	-0,47	0,00

APÊNDICE E - DADOS DAS PROPRIEDADES DE SOFTWARE

Tabela Apêndice E-1 - Dados Normalizados das Medidas

Características	Coesão			Acoplamento		
	OC	OA	MCA	OC	OA	MCA
Negrito	-0,65	-0,25	-0,31	0,11	-0,33	-0,19
Highlight	-0,12	-0,25	-0,31	-0,31	-0,27	-0,13
Enorme	-0,12	-0,12	-0,31	-0,63	-0,38	-0,41
OpenDyslexic	-0,12	-0,12	-0,31	-0,63	-0,64	-0,41
Tamanho	1,46	1,26	1,26	1,70	0,73	2,24
WebHelp	2,49	2,12	2,49	0,61	0,15	-0,53
Texto	1,49	1,26	1,58	0,80	0,60	1,13
Caracteres	-0,65	-0,78	-0,31	0,16	0,37	-0,19
Alinhamento	-0,65	-0,25	-0,31	0,11	-0,33	-0,24
Leitor	0,68	1,02	0,81	-0,36	-0,40	-0,36
Fonte	-1,17	-1,43	-1,47	0,90	1,21	1,19
OpenSans	-0,12	-0,12	-0,31	-0,63	-0,71	-0,41
Overlay	-0,12	-0,25	-0,31	-0,04	0,03	0,15
Linhas	-0,65	-0,78	-0,31	0,11	0,37	-0,19
Sublinhado	-0,65	-0,25	-0,31	0,11	-0,33	-0,19
Background	-0,65	-0,25	-0,31	0,27	-0,09	-0,13
Grande	-0,12	-0,12	-0,31	-0,63	-0,38	-0,41
Cor	1,58	1,58	1,3	0,76	0,41	1,00
Paragrafos	-0,65	-0,78	-0,31	0,16	0,37	-0,19
Regua	-0,12	-0,78	-0,31	-0,15	0,56	0,04
Pequeno	-0,12	-0,12	-0,31	-0,63	-0,38	-0,41
ComicSans	-0,12	-0,12	-0,31	-0,63	-0,64	-0,41
Georgia	-0,12	-0,12	-0,31	-0,63	-0,64	-0,41
Italico	-0,65	-0,25	-0,31	0,11	-0,33	-0,19
Medio	-0,12	-0,12	-0,31	-0,63	-0,38	-0,41