Manutenibilidade de Tecnologias para Programação de Linhas de Produtos de Software: Um Estudo Comparativo

José Natanael Reis[1], Gustavo Vale[2], Heitor Costa[2]

[1]Departamento Ciência da Computação - Universidade Federal de Lavras - Lavras - MG - Brasil

[2]Departamento de Ciência da Computação - Universidade Federal de Minas Gerais - Belo Horizonte - MG - Brasil

jnatanael@computacao.ufla.br, gustavovale@dcc.ufmg.br, heitor@dcc.ufla.br

Abstract. *In Software Product Line (SPL), the aim is a systematic and large-scale reuse. The SPL maintainability should be high, because change in a module can affect various products. Therefore, figuring out which technology/language that has greater maintainability can help software development companies, whose requires constant changes to choose the development technology. The main goal of this study is to assess the maintainability of three equivalent SPLs implemented with Feature-Oriented Programming using AHEAD, Aspect-Oriented Programming using AspectJ and Delta-Oriented Programming using DeltaJ. Six measures are used to evaluate four attributes: size; cohesion; coupling; and, complexity. As result, there is significant differences in the comparison only for one of the six measures, in which the maintainability of the SPLs developed in AspectJ and DeltaJ can be considered the same, but better than AHEAD SPL.*

Resumo. *No contexto de Linha de Produtos de Software (LPS), é visado o reúso sistemático e em larga escala. A manutenibilidade da LPS deve ser elevada, pois a mudança em um módulo pode impactar em vários produtos. Dessa forma, descobrir qual tecnologia/linguagem possui maior manutenibilidade, pode ajudar empresas desenvolvedoras de software, que necessitam de constantes alterações, a escolher a tecnologia de desenvolvimento. Neste trabalho, o objetivo é avaliar a manutenibilidade de três LPSs equivalentes implementadas com Programação Orientada a Características utilizando AHEAD, Programação Orientada a Aspectos utilizando AspectJ e com Programação Orientada a Delta utilizando DeltaJ. Um conjunto de seis medidas capazes de avaliar quatro atributos (tamanho, coesão, acoplamento e complexidade) foi utilizado na avaliação. Como resultados, houve diferença significativa na comparação, apenas para uma das seis medidas utilizadas no qual a manutenibilidade das LPSs desenvolvidas com AspectJ e DeltaJ são consideradas a mesma, no entanto, melhores que a LPS em AHEAD.*

1. Introdução

Um dos objetivos na Engenharia de Software é gerenciar e controlar a complexidade de sistemas de software [Sommerville, 2011]. Para atender as necessidades de clientes, os sistemas estão tornando-se cada vez mais complexos, o que dificulta o seu gerenciamento. Além disso, o reúso de (partes de) módulos de sistemas, quando não ordenados, pode comprometer a eficiência do desenvolvimento, tornando-o mais

custoso que o desenvolvimento a partir do zero [Pohl *et al*., 2005]. Para melhorar o gerenciamento de artefatos de código, principalmente artefatos reutilizados, algumas técnicas foram elaboradas, entre elas destaca-se a estratégia de desenvolvimento em Linha de Produtos de Software (LPS).

Uma LPS consiste em um conjunto de sistemas de software que possuem e compartilham características (*features*) comuns para satisfazer as necessidades de um domínio. Ela é desenvolvida a partir de um conjunto comum de ativos base de uma forma prescrita [Batory, 2003]. Uma LPS é fundamentada em uma estratégia para projeto e para implementação de sistemas, cujo objetivo é promover o reúso sistemático e em larga escala de características [Northrop, 2002]. Essas características permitem diferenciar os produtos presentes na LPS que apresentam funções comuns nos produtos e funções diferenciadas, conhecidas como variabilidade. Além disso, estudos comprovam os benefícios do uso dessa estratégia comparado com sistemas únicos [Pohl *et al*., 2005] e, a cada ano, mais estudos sobre LPSs são publicados [Vale *et al*., 2014].

Uma vez que o sistema pode ser decomposto em características, pode-se ter diferentes versões dele mediante a (re)combinação de certas características [Boucher *et al*., 2010]. Para o desenvolvimento de LPSs, podem ser utilizadas diferentes abordagens, por exemplo, tecnologias baseadas em composição ou tecnologias baseadas em anotação. Dentre as tecnologias baseadas em composição, podem ser citadas:

- Programação Orientada a Características (POC), utilizando AHEAD (*Algebraic Hierarchical Equations for Application Design*) [Batory, 2004];
- Programação Orientada a Aspectos (POA), utilizando AspectJ [Kiczales *et al*., 2001];
- Programação Orientada a Delta (POD), utilizando DeltaJ [Schaefer *et al*., 2011].

A estratégia de LPS difere do modelo tradicional de desenvolvimento de software, pois combina desenvolvimento genérico e desenvolvimento customizado, o que resulta em sistemas desenvolvidos em larga escala com a possibilidade de obter um produto customizado [Vale *et al*., 2015]. Além disso, com o alto índice de reúso e o gerenciamento preciso, custos e prazos podem ser reduzidos e o lucro pode ser aumentado. Como essa estratégia visa o reúso sistemático e em larga escala, a capacidade de manutenção da LPS deve ser elevada, pois, realizar manutenções em uma LPS, é mais complexo do que em sistemas únicos, a alteração em um módulo pode impactar vários produtos [Vale *et al*., 2015]. Assim, descobrir qual tecnologia/linguagem possui maior manutenibilidade pode ajudar empresas de software a escolher a tecnologia de desenvolvimento apropriada ao seu contexto.

Neste trabalho, o objetivo é avaliar a manutenibilidade de três LPSs equivalentes implementadas em AHEAD, AspectJ e DeltaJ. Para realizar essa avaliação foram utilizadas seis medidas que abrangem tamanho, coesão, acoplamento e complexidade. Essas medidas foram utilizadas em nível de componentes. Os componentes considerados são quaisquer arquivos que possuem código-fonte, tais como, constantes, refinamentos, aspectos e módulos delta. Para alcançar o objetivo, alguns procedimentos foram realizados, por exemplo, foram desenvolvidas duas LPSs (uma LPS em AspectJ e uma LPS em DeltaJ) equivalentes a LPS desenvolvida em AHEAD e o código-fonte das três LPSs foi normalizado. Os resultados do estudo mostraram que a LPS em AHEAD possui manutenibilidade inferior as LPSs em AspectJ e em DeltaJ e as LPSs em AspectJ e em DeltaJ possuem manutenibilidade equivalente.

O restante do artigo está organizado da seguinte forma. Conceitos importantes para o entendimento do trabalho estão resumidos na Seção 2. Alguns trabalhos relacionados são brevemente descritos na Seção 3. Os procedimentos metodológicos são apresentados na Seção 4. Os principais resultados e discussões são apresentados na Seção 5. Dificuldades no desenvolvimento das LPSs em AspectJ e em DeltaJ e lições aprendidas para contornar essas dificuldades foram descritas na Seção 6. Limitações e ameaças a validade do trabalho estão na Seção 7. Conclusões e sugestões de trabalhos futuros estão descritas na Seção 8.

2. Referencial Teórico

Nesta seção, são apresentados conceitos das três tecnologias para desenvolvimento de LPSs consideradas neste trabalho. Esses conceitos são importantes, pois destacam as peculiaridades de cada tecnologia com foco principal nas linguagens utilizadas. Além disso, é apresentado o conjunto de medidas utilizado para avaliar a manutenibilidade.

2.1. Programação Orientada a Características

Foi criada para síntese de sistemas de software em LPSs, no qual características são utilizadas para distinguir os sistemas de uma mesma família de produtos. Os produtos devem ser implementados em unidades de modularização sintaticamente independentes [Batory, 2003]. Um dos alicerces de POC é o refinamento sucessivo, que encapsula a implementação de uma característica. Esse conceito de refinamento sucessivo é simples e antigo na Engenharia de Software. Programas complexos devem ser desenvolvidos a partir de partes simples. Essas partes são as características do produto gerado a partir da LPS [Junior, 2008]. Os módulos criados em POC podem ser refinados de modo incremental, inserindo ou modificando classes (constantes e refinamentos), métodos, e atributos ou modificando a hierarquia de tipos. AHEAD é uma linguagem aceita e com popularidade na comunidade de POC.

2.2. Programação Orientada a Aspectos

Em POA, é oferecido suporte a mecanismos de abstração e de composição para alcançar melhor separação de interesses na implementação. Essa separação é possível com a introdução de nova unidade para a modularização dos interesses transversais, denominada aspecto [Kiczales *et al*., 2001]. Aspectos são utilizados para implementar interesses transversais capazes de entrecortar pontos bem definidos no fluxo de execução de um sistema de software [Junior, 2008]. AspectJ é uma linguagem de POA aceita e com popularidade na comunidade de desenvolvedores Java. A separação de interesses inclui o isolamento adequado, composição e reúso de código do aspecto [Kiczales *et al*., 1997].

2.3. Programação Orientada a Delta

O conceito principal de POD é módulo delta, recipiente de modificações para um sistema de software orientado a objetos, no qual utiliza a linguagem DeltaJ para implementação. Esse módulo especifica alterações no módulo central para implementar outros produtos. As alterações em um módulo delta podem ser em classe e na estrutura de classes. Modificar uma classe significa alterar a super classe para adicionar, modificar ou remover atributos ou métodos. A modificação de um método pode ser a

substituição do corpo do método por outra aplicação ou envolver o método existente utilizando a construção original [Schaefer *et al*., 2011].

2.4. Medição da Manutenibilidade de Software

Uma das dificuldades na Engenharia de Software é como medir a qualidade de um sistema de software [Sommerville, 2011]. Para ajudar a minimizar essa dificuldade, normas foram elaboradas, nas quais a qualidade de produtos é divida em características, dentre elas a manutenibilidade é explorada neste trabalho. Manutenibilidade de software é a facilidade com que um sistema ou componente de software é modificado para corrigir falhas ou melhorar o desempenho [ISO/IEC 25000, 2005]. Diversas maneiras para avaliar a manutenibilidade de um sistema foram propostas na literatura, por exemplo, índice de manutenibilidade, conjuntos de medidas e experimentos nos quais pessoas realizam a avaliação [Riaz *et al*., 2009].

A manutenibilidade de um sistema pode ser determinada como um todo, por componentes (por exemplo, classes e aspectos) ou por operações (por exemplo, métodos) [Bagheri; Gasevic, 2011]. Como o objetivo deste trabalho é comparar a manutenibilidade de diferentes tecnologias/linguagens, optou-se por trabalhar em nível de componentes. Um conjunto de seis medidas foi escolhido de forma a abranger os atributos de tamanho, de acoplamento, de coesão e de complexidade. As medidas escolhidas foram:

- Quantidade de Linhas de Código (*Lines of Code* - LOC);
- Falta de Coesão entre Métodos (*Lack of Cohesion of Methods* - LCOM);
- Profundidade da Árvore de Herança (*Depth of Inheriance Tree* - DIT);
- Acoplamento entre Objetos (*Coupling Between Objects* - CBO);
- Complexidade Ciclomática (*Cyclomatic Complexity* - CC);
- Métodos Ponderados por Classe (*Weighted Methods per Class* - WMC).

As duas primeiras medidas são de tamanho e de coesão, respectivamente. As duas medidas seguintes são de acoplamento e as duas últimas medidas são de complexidade. Essas medidas foram escolhidas por estarem entre as medidas mais citadas de uma Revisão Sistemática da Literatura [Riaz *et al*., 2009]. Além disso, com a escolha dessas medidas, procurou-se evitar quatro erros comuns ao utilizar medidas de software [Bouwers *et al*. 2012]:

- Usar uma medida sem interpretação apropriada;
- Fazer alterações apenas para melhorar o valor de uma medida;
- Utilizar uma única medida;
- Utilizar um conjunto amplo de medidas.

3. Trabalhos Relacionados

Nesta seção, são apresentados trabalhos relacionados às medidas escolhidas ou às propostas para avaliar a manutenibilidade de LPSs e trabalhos que comparam diferentes tecnologias para desenvolvimento de LPSs.

3.1 Medidas para Avaliar Manutenibilidade

Em um trabalho [Aldekoa *et al*., 2008], a medição da manutenibilidade de duas LPSs (`MUKalk` e `Graph Product Line`) orientadas a características foi realizada utilizando o índice de manutenibilidade (IM). Como procedimentos, o IM de cada

produto das LPSs foi calculado e, em seguida, a influência de cada característica foi verificada. Como principal resultado, foi proposto um modelo inicial para medir a manutenibilidade de uma LPS. Esse modelo é útil, principalmente, quando a variabilidade da LPS é baixa.

Em outro trabalho [Bagheri; Gasevic, 2011], o fato de diferentes produtos serem gerados a partir de uma LPS foi considerado. Com isso, pode ser previsto que um módulo de baixa qualidade tende a propagar-se em muitos produtos gerados. Portanto, indicadores precoces de atributos de qualidade são necessários para evitar a replicação de código defeituoso e de baixa qualidade. Assim, foi proposto um conjunto de 10 medidas estruturais direcionadas principalmente para modelos de características que abrangem complexidade, tamanho e acoplamento. Além disso, essas medidas são investigadas para detectar se são boas indicadoras de três subcaracterísticas de manutenibilidade (analisabilidade, instabilidade e compreensão). Com base no estudo, foi identificado um subconjunto de quatro medidas diretamente relacionadas com manutenibilidade, podendo ser utilizadas como indicadoras adequadas.

Uma Revisão Sistemática da Literatura [Abílio *et al*., 2012] sobre medidas relacionadas à manutenibilidade de sistemas de software desenvolvidos em tecnologias de POC e POA foi realizada. Nesse trabalho, observou-se a não-uniformidade na utilização dos termos "aspecto" e "característica" na definição ou na utilização das medidas. Isso pode levar usuários à confusão quanto qual medida utilizar. Como resultados principais, 111 medidas foram encontradas, sendo 33 medidas relacionadas à orientação a características e 78 medidas relacionadas à orientação a aspectos.

Com o conjunto de seis medidas utilizado neste trabalho, apresentado na Seção 2.4, procurou-se não cometer os quatro erros comuns [Bouwers *et al*., 2012] e abranger quatro atributos de qualidade de manutenibilidade: i) complexidade; ii) tamanho; iii) acoplamento; e iv) coesão (similar ao segundo trabalho, porém com adição da coesão). Com isso, mesmo conhecendo um conjunto extenso de medidas [Abílio *et al*., 2012; Bagheri; Gasevic, 2011], foi escolhido um conjunto reduzido para avaliar os quatro atributos de manutenibilidade tratados neste trabalho. Dentre o subconjunto apresentado no segundo trabalho [Bagheri; Gasevic, 2011], a única medida relacionada a código, complexidade ciclomática, está contida no conjunto de medidas utilizado neste trabalho.

3.2 Comparações entre Tecnologias de Programação

Uma Revisão Sistemática da Literatura [Ali *et al*., 2010] foi realizada para encontrar trabalhos que fizeram comparações entre sistemas POA e não POA. Nessa revisão, foram encontrados trabalhos que utilizavam linguagens de programação diferentes, por exemplo, Java, C, C++, AspectJ, EjFlow, CaesarJ e AspectC++. Para avaliação, foram considerados sete atributos (desempenho, tamanho de código, modularidade, capacidade de evolução, cognição e mecanismo da linguagem e tratamento de exceção). Esses atributos foram analisados em cada artigo selecionado na RSL e, quando existia uma avaliação a tecnologia superior, era pontuada. Como resultado da análise, a qualidade dos sistemas desenvolvidos em POA foi considerada elevada em relação aos não POA.

Em outro trabalho [Ferreira *et al*., 2013], foi realizado um estudo para comparar a modularidade e a propagação de mudanças de tecnologias para desenvolvimento de LPSs. Duas LPSs (`WebStore` e `MobileMedia`) desenvolvidas em três tecnologias diferentes (Compilação Condicional, Orientação Objetos com Padrões de Projeto e

POC) e 5 e 7 versões (`WebStore` e `MobileMedia`, respectivamente) para cada tecnologia. Essas versões podem ter componentes, métodos e linhas de código inseridas, alteradas ou removidas. Como resultados, POC teve desempenho ligeiramente melhor para propagação de mudanças e superior no quesito modularidade.

Outro estudo similar [Gaia *et al*., 2014] foi desenvolvido, porém com quantidade de versões e de tecnologias diferentes. Nesse estudo, foram utilizadas as tecnologias POC, POA, Compilação Condicional e Módulos de Características Aspectuais (MCA). Os resultados obtidos revelaram que a utilização de MCA e de POC tende a ser mais estável que as outras tecnologias. No entanto, POC não "lida bem" quando interesses transversais devem ser tratados.

Em um estudo prévio a este [Reis *et al*., 2014], foram avaliadas duas tecnologias para implementar LPS, POC (AHEAD) e POA (AspectJ). Nesse estudo, foram utilizadas cinco medidas para avaliar a manutenibilidade das LPSs, as quais avaliam tamanho, coesão, acoplamento e complexidade. Os resultados obtidos indicam que a `LPS TankWar` em POA (AspectJ) apresenta indícios que tenha maior manutenibilidade do que a `LPS TankWar` em POC (AHEAD).

Este trabalho difere-se dos demais por realizar a comparação das tecnologias utilizando LPSs equivalentes desenvolvidas com POC, POA e POD. Além disso, os trabalhos relacionados não possuem como objetivo principal avaliar a manutenibilidade de LPSs com foco no tamanho, na complexidade, no acoplamento e na coesão.

4. Procedimentos Metodológicos

Para cumprir o objetivo de avaliar a manutenibilidade de LPSs equivalentes desenvolvidas com tecnologias diferentes, foram propostas duas hipóteses (H_0 e H_1) e vários passos para conseguir obter um estudo sistemático e com viés minimizado. As hipóteses são:

- H_0: A manutenibilidade das LPSs em AHEAD, AspectJ e DeltaJ é equivalente;
- H_1: A manutenibilidade das LPSs em AHEAD, AspectJ e DeltaJ não é equivalente.

Em relação aos passos, inicialmente, tecnologias e linguagens para desenvolver LPSs com abordagem composicional foram estudadas. Por causa da quantidade de estudos encontrados, optou-se por estudar com profundidade POC, POA e POD, brevemente apresentados nas Seções 2.1, 2.2 e 2.3, respectivamente. Em seguida, procurou-se por LPSs para realizar um estudo comparativo. A `LPS TankWar` foi escolhida pelos motivos descritos na Seção 4.1. Essa LPS foi desenvolvida originalmente em AHEAD e, para realizar a comparação, foi necessário desenvolver versões em AspectJ e em DeltaJ.

Em seguida, foram pesquisadas formas de avaliar a manutenibilidade de LPSs. No entanto, como não foram encontrados estudos que se enquadrassem à proposta deste trabalho (abranger os quatro atributos de qualidade escolhidos), optou-se por montar um novo conjunto de medidas (Seção 2.4). Após esses passos, alguns procedimentos para medir o código-fonte das três LPSs foram realizados, eles estão descritos na Seção 4.2. Posteriormente, testes estatísticos foram realizados para comprovar se existem diferenças significativas entre os valores das medidas. Esses testes são brevemente descritos na Seção 4.3.

4.1. `LPS TankWar`

A `LPS TankWar` é um jogo de tanques de guerra originalmente desenvolvido em AHEAD por estudantes da Universidade de Magdeburg. Essa LPS pode ser executada em computadores ou telefones celulares [Schulze *et al*., 2012]. Uma caracterização da `LPS TankWar` foi realizada para conhecê-la e auxiliar na comparação com as novas versões a serem geradas (Tabela 1). Essa LPS possui 5.640 linhas no total, sendo que 86,3% de código, 1,9% comentários de documentação e 11,9% de linhas gerais e em branco.

Tabela 1 - Informações sobre a LPS `TankWar`

Medidas		Quantidade
Linhas de	Código	4.865
	Comentários de Documentação	44
	Comentários Gerais	61
Classes (Constantes)		29
Interfaces (Constantes)		1
Refinamentos de Classes		56
Características	Abstratas	6
	Concretas	31
Produtos de Software		39.060
Classes		85

Entre as variações dos produtos dessa LPS estão idioma (Inglês ou Alemão), nível de dificuldade (fácil ou difícil), escolha da quantidade de tipos de tanque para escolha do usuário (1, 2 ou 3), escolha de ferramentas para incorporar ao tanque (de 0 a 6) e possibilidade de armazenar resultados em um *ranking*. Ela foi escolhida por causa do seu tamanho e por ter sido utilizada em outros estudos [Apel; Beyer, 2011; Schulze *et al*., 2012; Schulze *et al*., 2010]. A `LPS TankWar` possui modelo de características com 37 características (6 abstratas e 31 concretas) [Schulze *et al*., 2012], podendo gerar 39.060 produtos, e possui 85 classes (29 constantes e 56 refinamentos).

4.2. Procedimentos para Medição

Com a finalidade de deixar o código das LPSs "mais parecidos" e tornar a comparação entre as medidas mais coerente, foi realizada normalização no código. Essa normalização seguiu os padrões de codificação Java, pois as linguagens de programação em estudo são baseadas em Java. Os padrões de codificação Java foram aplicados com auxílio da documentação presente no Google Java Style[1].

No processo de medição do código, foram utilizadas duas heurísticas para fazer a conversão dos arquivos originais para arquivos Java. Essa conversão foi necessária por causa da falta de *plug-ins* para medir sistemas de software nas linguagens utilizadas, podendo medir o código incorretamente se fosse utilizado um *plug-in* inapropriado. Para cada componente, foi criado um arquivo Java para representá-lo. Nessa conversão, foram utilizadas as seguintes heurísticas:

- Trocar a extensão do arquivo `.jak`, `.aj` e `.dj` para `.java`;
- Modificar algumas palavras chave de cada tecnologia, por exemplo, trocar `refines` (AHEAD), `aspect` (AspectJ) e `core` (DeltaJ) por `class`.

[1] http://google-styleguide.googlecode.com/svn/trunk/javaguide.html

Essas heurísticas foram utilizadas para evitar modificações maiores no código que poderiam, por exemplo, afetar a medição das LPSs introduzindo particularidades de Java ou removendo particularidades das demais linguagens. Os *plug-ins* `VizzAnalyzer`[2] e `Code Pro Analytix`[3] foram utilizados para medir as LPSs. Com o `VizzAnalyzer`, foi coletado o valor das medidas LOC, LCOM, DIT, CBO e WMC e com o `Code Pro Analytix` foi coletado o valor da medida CC.

4.3. Testes Estatísticos

Para fornecer resultados robustos na comparação da manutenibilidade das três LPSs e aceitar uma das hipóteses, foi utilizado o processo de combinação. As combinações foram:

- `LPS TankWar` em AHEAD x `LPS TankWar` em AspectJ;
- `LPS TankWar` em AHEAD x `LPS TankWar` em DeltaJ;
- `LPS TankWar` em AspectJ x `LPS TankWar` em DeltaJ.

O teste de Scott e Knott foi escolhido para analisar as combinações. Para isso, foi necessário realizar previamente a análise de variância e o teste *t-student*. A análise de variância é uma técnica que permite decompor a variação total observada nos dados experimentais em causas conhecidas e não conhecidas. O teste *t-student* é utilizado para comparar as médias dos componentes, no qual o valor de `t` é comparado com o valor de `F` calculado na análise de variância [Ferreira, 2009]. O teste de Scott e Knott é a razão da verossimilhança para testar a significância de que `n` tratamentos (neste trabalho, entende-se por componentes) podem ser divididos em dois grupos que maximizem a soma de quadrados entre grupos [Ferreira, 2009].

Apenas o teste *t-student* poderia ser utilizado, mas tal comparação não seria consistente. O principal motivo para escolher o teste de Scott e Knott é a ausência de ambiguidade nos procedimentos de comparações múltiplas. Os testes de hipótese foram realizados com nível de confiança igual a 99%.

5. Resultados e Discussão

Na Tabela 2, são apresentados os resultados obtidos com a medição das LPSs. Na primeira coluna, estão os componentes em ordem alfabética. Nas demais colunas, está o valor das medidas. Para facilitar a comparação, cada medida da `LPS TankWar` em AHEAD foi disposta ao lado da medida correspondente da `LPS TankWar` em AspectJ e da `LPS TankWar` em DeltaJ. Por exemplo, na primeira linha, o componente `Beschleunigung` possui valores para as medidas LOC, LCOM, DIT, CBO, WMC e CC, respectivamente:

- `69`; `17`; `0`; `2`; `11` e `2,6` na `LPS TankWar` em AHEAD;
- `50`; `7`; `0`; `1`; `6` e `2,66` na `LPS TankWar` em AspectJ;
- `62`; `0`; `0`; `0`; `0` e `1` na `LPS TankWar` em DeltaJ.

Nessa tabela, o valor "0" é um indicativo obtido na medida e o símbolo "--" é um indicativo de que o componente não pertence àquela tecnologia, por exemplo, apenas a `LPS TankWar` em DeltaJ possui o componente `Core`.

[2] http://www.arisa.se/tools.php?lang=en
[3] https://developers.google.com/java-dev-tools/codepro/doc/?hl=pt-BR

Tabela 2 - Medidas dos Componentes das LPSs em AHEAD, AspectJ e em DeltaJ

Componente	LOC 1	LOC 2	LOC 3	LCOM 1	LCOM 2	LCOM 3	DIT 1	DIT 2	DIT 3	CBO 1	CBO 2	CBO 3	WMC 1	WMC 2	WMC 3	CC 1	CC 2	CC 3
Beschleunigung	69	50	62	17	7	0	0	0	0	2	1	0	11	6	0	2,6	2,66	1
Bombe	61	46	61	16	1	0	0	0	0	3	0	0	10	1	0	3	1	1
ChinaType99	13	10	16	1	1	0	0	0	0	3	0	0	1	1	0	1	1	1
Core	--	--	5	--	--	1	--	--	0	--	--	1	--	--	1	--	--	1
DE	25	19	21	0	1	0	0	0	0	2	1	0	0	1	0	0	1	1
Dtank	--	--	19	--	--	1	--	--	0	--	--	1	--	--	1	--	--	4,17
Dtool	--	--	48	--	--	4	--	--	0	--	--	0	--	--	7	--	--	2,16
Easy	22	20	25	1	1	0	0	0	0	2	0	3	3	1	0	3	1	1
einfrieren	67	57	67	17	7	0	0	0	0	1	0	1	14	8	0	3,39	3,33	1,87
EN	24	19	20	0	1	0	0	0	0	2	1	2	0	1	0	0	1	1
Energie	39	31	43	16	1	0	0	0	0	1	1	6	6	1	0	1,5	1	1
explodieren	30	23	29	9	1	0	0	0	0	1	2	2	4	1	0	1,33	1	1
ExplodierenEffekt	160	146	146	8	8	8	1	1	1	1	0	0	13	13	13	3,6	3,6	3,6
Feuerkraft	60	50	58	17	1	0	0	0	0	2	2	0	11	1	0	2,6	1	1
fuerHandy	351	351	356	4	4	1	0	0	0	3	2	3	103	2	1	52	1	1
fuerPC	352	354	358	4	2	0	0	0	0	6	3	0	103	2	1	52	1	1
GameManager	32	32	32	49	49	49	0	0	0	6	0	0	7	7	7	1	1	1
GameObject	113	147	147	31	79	79	0	0	0	2	0	0	16	22	22	2,77	2,06	2,06
GermanyLeopard	13	11	15	1	1	0	0	0	0	0	1	3	1	1	0	1	1	1
Handy	385	70	75	1003	16	1	0	0	0	0	0	0	66	4	1	1,83	1	1
Hard	56	57	60	1	1	0	0	0	0	0	0	1	12	1	0	16	1	1
Image	163	163	170	23	7	0	0	0	0	7	2	0	24	4	2	9,4	2,33	5
IMGtool	28	30	32	1	1	0	0	0	0	0	0	2	2	1	0	7	1	1
InfoPanel	43	44	44	0	0	0	0	0	0	0	0	2	4	4	4	3	3	3
KeyMonitor	73	14	14	1	9	9	0	0	0	1	1	1	5	3	3	7	1	1
M240	10	8	12	1	1	0	0	0	0	0	1	0	1	1	0	1	1	1
M600	12	8	12	1	1	0	0	0	0	0	0	0	1	1	0	1	1	1
M780	10	8	12	1	1	0	0	0	0	0	0	0	1	1	0	1	1	1
Maler_Handy	349	--	--	0	--	--	0	--	--	1	--	--	55	--	--	1,87	--	--
Maler_PC	349	--	--	0	--	--	0	--	--	0	--	--	55	--	--	1,87	--	--
Maler	--	385	385	--	127	127	--	0	0	--	0	1	--	63	63	--	1,69	1,69
MalerZeit	21	21	21	0	0	0	0	0	0	0	2	0	2	2	2	2	2	2
MapInfo	133	134	134	0	0	0	0	0	0	4	3	0	13	13	13	7,5	7,5	7,5
Mars	61	56	61	18	23	0	0	0	0	2	0	3	11	7	0	2	1,6	1
Menu_Handy	313	--	--	48	--	--	0	--	--	1	--	--	55	--	--	5,15	--	--
Menu_PC	313	--	--	48	--	--	0	--	--	0	--	--	55	--	--	5,15	--	--
Menu	--	318	318	--	48	48	--	0	0	--	2	0	--	55	55	--	5,15	5,15
MIDlet	3	3	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Missile	160	219	219	16	272	272	1	1	1	1	2	0	17	32	32	4,28	2,14	2,14
NameChar	25	25	25	0	0	0	0	0	0	1	3	3	5	5	5	1,5	1,5	1,5
NameTextField	118	121	121	3	3	3	0	0	0	1	0	0	19	19	19	2,5	2,5	2,5
Note_Handy	17	17	17	0	0	0	0	0	0	2	0	0	0	0	0	3	3	3
Note_PC	17	17	17	0	0	0	0	0	0	3	0	0	0	0	0	3	3	3
Option_Handy	33	--	--	0	--	--	0	--	--	2	--	--	0	--	--	1	--	--
Option_PC	33	--	--	0	--	--	0	--	--	2	--	--	0	--	--	1	--	--
Option	--	33	33	--	0	0	--	0	0	--	2	1	--	0	0	--	1	1
PC	70	16	15	10	1	0	0	0	0	0	0	1	9	1	0	1,66	1	1
Record	48	43	46	12	1	0	0	0	0	2	0	7	8	1	0	1,5	1	1
Record_Handy	103	103	103	0	0	0	0	0	0	0	3	2	11	11	11	8	8	8
Record_PC	105	105	105	0	0	0	0	0	0	2	2	1	11	11	11	14	14	14
ReFuerHandy	112	110	104	32	17	2	0	0	0	1	2	2	19	10	5	3,25	3,39	1,6
ReFuerPC	86	85	81	28	7	2	0	0	0	2	0	2	14	6	5	2,33	2	1,6
Soundeffekt	3	3	3	0	0	0	0	0	0	1	0	2	0	0	0	0	0	0
SoundFuerHandy	60	57	55	23	14	0	0	0	0	3	2	0	11	7	0	3,2	1,75	1
SoundFuerPC	83	79	76	23	1	0	0	0	0	0	1	0	11	1	0	4,2	1	1
SoundPlayer_Handy	138	138	138	142	142	142	0	0	0	0	0	2	30	30	30	2,69	2,69	2,69
SoundPlayer_PC	63	63	63	0	0	0	0	0	0	5	2	0	11	11	11	1,1	1,1	1,1
Sprach	4	128	128	0	572	572	0	0	0	1	7	1	0	26	26	0	1	1
Tank	467	524	524	91	487	487	1	1	1	1	0	3	57	71	71	7,35	4,17	4,17
TankManeger	239	257	257	4	48	48	1	1	1	0	3	0	33	35	35	3,06	2,82	2,82
Tool	49	51	51	15	26	26	1	1	1	0	6	0	9	10	10	2,16	2,16	2,16
Tools	64	67	71	49	47	14	0	0	0	1	1	0	14	9	6	2,28	1,42	1,75
USAM1Abrams	12	11	14	1	1	0	0	0	0	3	3	0	1	1	0	1	1	1
Wall	102	102	102	0	0	0	1	1	1	1	1	0	12	12	12	3,6	3,6	3,6

Os valores "1", "2" e "3" após a sigla das medidas correspondem à `LPS TankWar` em AHEAD, `LPS TankWar` em AspectJ e `LPS TankWar` em DeltaJ, respectivamente.

A sumarização da quantidade de componentes das LPSs e a média da quantidade de linhas totais dos componentes são apresentadas na Tabela 3. As LPSs possuem entre 55 e 58 componentes e médias das linhas totais (código + comentários + documentação) dos componentes entre 90 e 103. A `LPS TankWar` em DeltaJ possui componentes menores, em termos de linhas totais, do que as demais LPSs (90,5 linhas) e a `LPS TankWar` em AspectJ possui quantidade menor de componentes (55 componentes).

Tabela 3 - Sumarização do Tamanho das LPSs

LPS		Quantidade de Componentes	Média da Quantidade de Linhas Totais por Componente
TankWar	AHEAD	58	102,83
	AsptecJ	55	91,98
	DeltaJ	58	90,5

Em relação aos componentes e às linhas de código totais, pode-se dizer que nem sempre uma LPS com maior quantidade de componentes e com menor quantidade de linhas na média representa uma LPS com melhor manutenibilidade. Pois, se os interesses presentes nesses componentes não estiverem bem modularizados, não adianta ter essas quantidades. Isso, reforça os quatro erros comuns ao utilizar medidas de software [Bouwers *et al*., 2012]. A ideia central é apresentar um esboço das LPSs. Em seguida, são apresentados os resultados dos testes estatísticos realizados para cada medida (Figura 1). Pode-se observar que em nenhum caso o valor de `F` calculado (`Fc`) é significativo, pois o valor da verificação de `F` calculado é menor ou maior que o `F` tabelado (`Pr`) e é maior que o nível de significância (`0,01`). Por exemplo, no caso de LOC, o valor de `Fc` é igual a `0,209` enquanto o valor de `Pr` é `0,08113`.

Análise da Variância para a Medida LOC

FV	GL	SQ	QM	Fc	Pr
Linguagem	2	5.224,9792	2.612,4896	0,209	0,8113
Erro	168	2.096.673,7577	12.480,2009		
Total Corrigido	170	2.101.898,7368			

Análise da Variância para a Medida LCOM

FV	GL	SQ	QM	Fc	Pr
Linguagem	2	1.164,7809	582,3905	0,044	0,9573
Erro	168	2.239.649,6401	13.331,2478		
Total Corrigido	170	2.240.814,4210			

Análise da Variância para a Medida DIT

FV	GL	SQ	QM	Fc	Pr
Linguagem	2	0,0012	0,0006	0,006	0,9938
Erro	168	16,1041	0,0958		
Total Corrigido	170	16.1053			

Análise da Variância para a Medida CBO

FV	GL	SQ	QM	Fc	Pr
Linguagem	2	7,5223	3,7611	1,597	0,2056
Erro	168	395,6473	2,3550		
Total Corrigido	170	403,1696			

FV: Fator de Variação; **GL**: Grau de Liberdade; **SQ**: Soma de Quadrados; **QM**: Quadrado Médio; **Fc**: Valor de F calculado; **Pr**: Verificação do Fc é menor ou maior que o F tabelado

Figura 1 - Análise de Variância para as Medidas

Análise da Variância para a Medida WMC

FV	GL	SQ	QM	Fc	Pr
Linguagem	2	2.281,9662	1.140,9831	3,297	0,0394
Erro	168	58.135,5542	346,0449		
Total Corrigido	170	60.417,5205			

Análise da Variância para a Medida CC

FV	GL	SQ	QM	Fc	Pr
Linguagem	2	280,9493	140,4747	4,136	0,0176
Erro	168	5.706,3021	33,9661		
Total Corrigido	170	5.987,2515			

FV: Fator de Variação; **GL**: Grau de Liberdade; **SQ**: Soma de Quadrados; **QM**: Quadrado Médio; **Fc**: Valor de F calculado; **Pr**: Verificação do Fc é menor ou maior que o F tabelado

Figura 1 - Análise de Variância para as Medidas (cont.)

Na Figura 2, são apresentados os resultados do teste de Scott e Knott para cada medida utilizada neste estudo. A existência de diferença significativa é representada pela alternância de letras. Como pode ser observado, não existe diferença significativa para as medidas LOC, LCOM, DIT, CBO e WMC, porém há diferença significativa para a medida CC. Por exemplo, para a medida LOC, o resultado é igual a A para todas as LPSs e, para a medida CC, o resultado é igual a A para as `LPSs TankWar` em AspectJ e em DeltaJ e igual a B para a `LPS TankWar` em AHEAD. Isso significa que, no segundo caso, a manutenibilidade da `LPS TankWar` em AspectJ e da `LPS TankWar` em DeltaJ são equivalentes e a manutenibilidade da `LPS TankWar` em AHEAD é menor do que das outras duas LPSs.

Teste de *Scott e Knott* para a Medida LOC

Linguagem	Médias	Resultado do Teste
DeltaJ	1.0172	A (0,2033)
AspectJ	1.1818	A (0,2033)
AHEAD	1.5172	A (0,2033)

Teste de *Scott e Knott* para a Medida LCOM

Linguagem	Médias	Resultado do Teste
AHEAD	30.7931	A (15,2979)
DeltaJ	32.6896	A (15,2979)
AspectJ	37.0727	A (15,2979)

Teste de *Scott e Knott* para a Medida DIT

Linguagem	Médias	Resultado do Teste
DeltaJ	0,1034	A (0,041)
AHEAD	0,1034	A (0,041)
AspectJ	0,1091	A (0,041)

Teste de *Scott e Knott* para a Medida CBO

Linguagem	Médias	Resultado do Teste
DeltaJ	1.0172	A (0,2033)
AspectJ	1.1818	A (0,2033)
AHEAD	1.5172	A (0,2033)

Teste de *Scott e Knott* para a Medida CC

Linguagem	Médias	Resultado do Teste
DeltaJ	2,1177	A (0,7722)
AspectJ	2.1665	A (0,7722)
AHEAD	4.8486	B (0,7722)

Teste de *Scott e Knott* para a Medida WMC

Linguagem	Médias	Resultado do Teste
DeltaJ	8.3621	A (2,4647)
AspectJ	9.7818	A (2,4647)
AHEAD	16.6724	A (2,4647)

Valor entre parênteses corresponde ao erro padrão da média

Figura 2 - Resultados do Teste de Scott e Knott para as Medidas

Com a análise de variância, não foi encontrada diferença significativa, com um nível do confiança de 99%, para as medidas. No entanto, com o teste de Scott e Knott, apenas para a medida CC, foi encontrada diferença significativa. Quando o valor das médias do teste de Scott e Knott é comparado, percebe-se que em todos os casos as médias da `LPS TankWar` em AHEAD são maiores em relação as médias das outras LPSs. Isso reforça levemente o resultado obtido com a medida CC, mesmo não representando diferença significativa. Isso implica que a manutenibilidade da `LPS TankWar` em AspectJ e da `LPS TankWar` em DeltaJ são iguais e melhores que a manutenibilidade da `LPS TankWar` em AHEAD. Considerando esse resultado, a

hipótese H_0 é rejeitada e a hipótese H_1 é aceita, pois a manutenibilidade das três LPSs é diferente.

Em comparação com resultados de alguns trabalhos relacionados (Seção 2.2), LPSs desenvolvidas utilizando POC geralmente apresentam resultados melhores ou equivalentes em relação a outras tecnologias. No entanto, neste trabalho, POC pode não ter tido um resultado positivo em relação às demais LPSs pelos motivos apresentados na Seção 7 e pela perspectiva na qual essa avaliação foi realizada neste trabalho.

6. Dificuldades e Lições Aprendidas no Desenvolvimento das LPSs

Como a `LPS TankWar` encontrada foi desenvolvida originalmente em AHEAD, apenas as LPSs equivalentes em AspectJ e em DeltaJ foram desenvolvidas para este trabalho. A versão original (AHEAD) serviu de base para o desenvolvimento das novas versões (AspectJ e DeltaJ) e, quando possível, partes de código em AHEAD foram reutilizados para as novas versões da `LPS TankWar`. Para manter mesmo estilo de programação e a `LPS TankWar` em AHEAD não ter desvantagens ou vantagens, uma normalização no código foi realizada (Seção 4.2).

As três linguagens utilizadas neste trabalho são baseadas em Java. O desenvolvimento das LPSs em AspectJ e em DeltaJ diferenciam-se do desenvolvimento da LPS em AHEAD, pois tratam-se de tecnologias complementares a orientação a objetos. Dessa forma, essas duas linguagens utilizam código puramente Java em longos trechos de código (desenvolvimento das partes comuns) e pontos específicos das linguagens, como aspectos e módulos delta, fornecem variabilidade para a LPS.

As principais dificuldades encontradas na implementação da `LPS TankWar` em AspectJ foram: i) entender como seriam os pontos de junção de cada aspecto referente a `LPS TankWar` em AHEAD; e ii) entender como implementar os refinamentos da `LPS TankWar` em AHEAD para a `LPS TankWar` em AspectJ. Na implementação, algumas conversões foram diretas, pois algumas características não precisavam de refinamentos. Para contornar tais dificuldades, cada característica foi estudada separadamente para conversão em aspectos, por exemplo, os pontos de junção foram aqueles em que a característica refinada dependia de outro refinamento e cada refinamento foi implementado como um aspecto diferente. Dessa forma, refinamentos com código duplicado foram descartados.

As principais dificuldades encontradas na implementação a `LPS TankWar` em DeltaJ foram: i) dividir os módulos delta da `LPS TankWar` em AHEAD; ii) gerar o módulo central; e iii) separar cada característica em módulos delta. Seguindo o mesmo princípio utilizado na `LPS TankWar` em AspectJ, algumas das características foram convertidas diretamente de AHEAD para DeltaJ. Para contornar tais dificuldades: i) características foram separadas entre básicas e de personalização da LPS; ii) um módulo central com as características básicas da LPS foi implementado; e iii) cada característica de personalização foi separada em um módulo delta.

7. Ameaças à Validade

Mesmo com cuidadoso planejamento do estudo, alguns fatores podem ser considerados na avaliação da validade dos resultados. Primeiro, um conjunto de medidas foi escolhido para avaliar a manutenibilidade. Esse conjunto foi escolhido porque

representam quatro atributos de qualidade considerados importantes quando se deseja avaliar a manutenibildiade de um sistema de software. Talvez, se outras medidas forem utilizadas, resultados diferentes poderiam ter sido encontrados. Em segundo lugar, a manutenibilidade foi tratada em termos de quatro atributos de qualidade; no entanto, é sabido que manutenibilidade é complexa de medir e outros atributos podem ser utilizados, tais como, modularização dos componentes e propagação de mudanças. É possível que algum erro no cálculo das medidas ou da análise estatística tenha ocorrido. De forma a tentar minimizar essa possibilidade, dois pesquisadores computaram o valor das medidas e realizaram o teste de Scott e Knott em computadores diferentes. Além disso, foi utilizado como padrão o nível de confiança de 99% para realizar esse teste. Outra porcentagem poderia ser escolhida e um resultado diferente poderia ter sido obtido.

Não foram encontradas ferramentas computacionais para medir código das LPSs utilizando as medidas utilizadas neste trabalho nem para quaisquer outras medidas (uma limitação). Embora, seja sabido que utilizar heurísticas para transformar arquivos dessas linguagens (`.jak`, `.aj` e `.dj`) em arquivos Java (`.java`) sem perder a sintaxe não é o caminho ideal. Essa foi uma alternativa mais direta utilizada para medir o código das LPSs de maneira automática, imparcial e objetiva. A `LPS TankWar` pode não representar todas as propriedades de sistemas do mundo real. Apenas linguagens de programação baseadas em Java foram utilizadas; talvez, realizando a comparação com outras linguagens não baseadas em Java, tais como, AspectC++ e FeatureC++, poderiam tornar a comparação mais complexa, pois as medidas poderiam sofrer variações e diferentes tipos de construtores poderiam ser utilizados e os resultados poderiam ser diferentes.

Para a avaliação da manutenibilidade entre as tecnologias (POC, POA e POD), foi realizado um estudo comparativo. Se mais estudos comparativos, com mais LPSs, fossem realizados, os resultados poderiam ser diferentes e/ou menos árduos de serem generalizados. O fato de utilizar o código da `LPS TankWar` em AHEAD para o desenvolvimento da `LPS TankWar` em AspectJ e da `LPS TankWar` em DeltaJ pode ter influenciado o resultado. Pois, o código dessa LPS foi minuciosamente analisado e maneiras mais eficientes para resolver a mesma tarefa podem ter sido utilizadas implicitamente para as novas versões em AspectJ e em DeltaJ. Além disso, como há diferença entre os desenvolvedores da `LPS TankWar` em AHEAD e da `LPS TankWar` em AspectJ e da `LPS TankWar` em DeltaJ, estilos/padrões de codificação diferentes podem ter sido utilizados. Para tentar minimizar esse diferença, o código das LPSs foi normalizado utilizando padrões de codificação Java.

8. Conclusões

Dada a necessidade de conhecer a manutenibilidade de LPSs e qual tecnologia pode ser mais manutenível, foi realizada uma comparação da manutenibilidade de três LPSs equivalentes desenvolvidas em AHEAD, em AspectJ e em DeltaJ. Para alcançar esse objetivo, foi necessário encontrar um LPS, desenvolver versões dessa LPS nas demais linguagens para ter três LPSs equivalentes desenvolvidas, escolher um conjunto de medidas para representar a manutenibilidade de um sistema de software, medir essas LPSs e fornecem evidências de qual tecnologia é mais manutenível.

Os resultados mostraram que para cinco das seis medidas utilizadas (LOC, LCOM, DIT, CBO e WMC) não há diferença significativa na manutenibilidade das LPSs. Uma diferença significativa foi encontrada apenas para a medida CC, na qual a `LPS TankWar` em AspectJ e a `LPS TankWar` em DeltaJ possuem manutenibilidade equivalentes e a `LPS TankWar` em AHEAD possui manutenibilidade inferior a elas. Mesmo com esse resultado, é importante destacar que cada tecnologia possui seus pontos fortes para modularizar interesses e representar a variabilidade de LPSs.

Este trabalho pode contribuir com um conjunto de medidas que se mostrou coerente e eficaz para medir a manutenibilidade de LPSs. Esse conjunto de medidas abrange quatro atributos (tamanho, acoplamento, coesão e complexidade). Além disso, ele apresenta caminhos para contornar dificuldades no desenvolvimento de LPS em AspectJ e de LPS em DeltaJ, heurísticas para conseguir medir código em AHEAD, em AspectJ e em DeltaJ com *plug-ins* desenvolvidos para medir código Java e os resultados de uma comparação entre três tecnologias/linguagens diferentes.

Como sugestões para trabalhos futuros, o estudo com outras linguagens e/ou tecnologias de programação pode ser repetido, a realização de um estudo similar com outras LPSs, a verificação dos resultados serem replicáveis para outros domínios, a ampliação do conjunto de medidas utilizadas para medir a manutenibilidade de LPSs, o desenvolvimento de uma ferramenta para medir código de LPSs em DeltaJ, em AspectJ e em AHEAD e a criação de uma extensão para o *plug-in* FeatureIDE para efetuar análise da manutenibilidade de LPSs.

Agradecimentos

Nós agradecemos CNPq e CAPES pelo apoio financeiro para realizar esse trabalho.

Referências

Abílio, R.; Teles, P.; Costa, H.; Figueiredo, E. (2012) A Systematic Review of Contemporary Metrics for Software Maintainability. In: Brazilian Symposium on Software Components, Architecture and Reuse, pp.130-139.

Aldekoa, G.; Trujillo, S.; Sagardui, G.; Diaz, O. (2008) Quantifying Maintainability in Feature Oriented Product Lines. In: European Confon Software Maintenance and Reengineering, pp. 243-247.

Ali, M.; Babar, M. A.; Chen, L.; Stol, K.-J. (2010) A Systematic Review of Comparative Evidence of Aspect-Oriented Programming. In: Journal Information and Software Technology, v.52, n.9, pp.871-887.

Apel, S.; Beyer, D. (2011) Feature Cohesion in Software Product Lines: An Exploratory Study. In: International Conference on Software Engineering, pp.421-430.

Bagheri, E.; Gasevic, D. (2011) Assessing the Maintainability of Software Product Line Feature Models using Structural Metrics. In: Software Quality Journal, v. 19, n. 3, pp. 579-612.

Batory, D. (2003) A Tutorial on Feature Oriented Programming and Product-Lines. In: International Conference on Software Engineering. p. 753.

Batory, D. (2004) Feature-Oriented Programming and the AHEAD Tool Suite. In: 26th International Conference on Software Engineering, pp. 702-703, 2004.

Boucher, Q.; Classen, A.; Faber, P.; Heymans, P. (2010) Introducing TVL, a Text-Based Feature Modelling Language. In: International Workshop on Variability Modelling of Software-intensive Systems. pp. 159-162.

Bouwers, E.; Visser, J.; Deursen, A. (2012) Getting what you measure. Communications of the ACM, vol. 55, no. 7, pp. 54-59.

Ferreira, D. F. (2009) Estatística Básica. Ed. Ufla, 2ª Ed, 664p.

Ferreira, G. ; Gaia, F. N. ; Figueiredo, Eduardo ; Maia, Marcelo. (2013) On the use of feature-oriented programming for evolving software product lines - A comparative study. Science of Computer Programming, v. 72, p. 1-34.

Gaia, F.; Ferreira, G.; Figueiredo, E.; Maia, M. (2014) A quantitative and qualitative assessment of aspectual feature modules for evolving software product lines. Science of Computer Programming. v. 1, p. 1-36.

ISO/IEC 25000 (2005) "Software Engineering - Software Product Quality requirements and Evaluation (SQuaRE)" - Guide to SQuaRE, p. 35.

Junior, C. A. de F. P. (2008) Geração de Aplicações para Linhas de Produtos Orientadas a Aspectos com Apoio da Ferramenta Captor-AO. Dissertação de Mestrado em Ciências de Computação e Matemática Computacional - Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 120p.

Kiczales, G.; Hilsdale, E.; Hugunin, J.; Kersten, M.; Palm, J.; Griswold, W. G. (2001) An Overview of Aspect. In: European Conference on Object-Oriented Programming. pp. 327-353.

Kiczales, G.; Lamping, J.; Mendhekar, A.; Maeda, C.; Lopes, C.; Loingtier, J. M.; Irwin, J. (1997) Aspect-Oriented Programming. In: ECOOP. pp. 220-242.

Northrop, L. (2002) Sei's Software Product Line Tenets. In: IEEE Software, v. 19, n. 4, pp. 32-40.

Pohl, K.; Bockle, G.; Linden, F. J. (2005) Software Product Line Engineering: Foundations, Principles, and Techniques. Springer-Verlag, 494p., 2005.

Reis, J. N.; Vale, G.; Costa, H. (2014) Análise Comparativa de Tecnologias de programação no Contexto de Linhas de Produtos de Software. Simpósio Mineiro de Engenharia de Software, p. 167 - 176.

Riaz, M.; Mendes, E.; Tempero, E. (2009) A Systematic Review of Software Maintainability Prediction and Metrics. In: International Symposium Empirical Software Engineering and Measurement, p. 367-377.

Schaefer, I.; Bettini, L.; Damiani, F. (2011) Compositional Type Checking for Delta-Oriented Programming. In: International Conference on Aspect-oriented Software Development. pp. 43-56.

Schulze, S.; Apel, S.; Kastner, C. (2010) Code Clones in Feature-Oriented Software Product Lines. In: International Conference on Generative Programming, pp.103-112.

Schulze, S.; Thüm, T.; Kuhlemann, M.; Saake, G. (2012) Variant-Preserving Refactoring in Feature-Oriented Software Product Lines. In: Workshop on Variability Modeling of Software-Intensive Systems, pp.73-81.

Sommerville, I. (2011) Engenharia de Software, 9ª Edição, Pearson Education.

Vale, G.; Abílio, R.; Freire, A.; Costa, H. (2015) Criteria and Guidelines to Improve Software Maintainability. In Software Product Lines. International Conference on Information Technology: New Generations.

Vale, G.; Figueiredo, E.; Abílio, R.; Costa, H. (2014) Bad Smells in Software Product Lines: A Systematic Review. In: Eighth Brazilian Symposium on Software Components, Architectures and Reuse, pp. 84-93.