Association for Information Systems
AIS Electronic Library (AISeL)

Proceedings of the XI Brazilian Symposium on Information Systems (SBSI 2015)

Brazilian Symposium on Information Systems (SBIS)

5-2015

A Systematic Study on Approaches to deal with the Systems' Evolution and Customization

Fernanda Almeida Passos
Universidade Federal de Sergipe, fernanda@ufs.br

Kleber T. O. Santos
Universidade Federal de Sergipe, klebertarcisio@yahoo.com.br

Raphael F. S. Barreto
Universidade Federal de Sergipe, rfsbarreto@gmail.com

Galileu Santos de Jesus
Universidade Federal de Sergipe, galilasmb@gmail.com

Glayderson Jonathan Nunes
Universidade Federal de Sergipe, glaydersonjonathan@gmail.com

See next page for additional authors

Authors
Fernanda Almeida Passos, Kleber T. O. Santos, Raphael F. S. Barreto, Galileu Santos de Jesus, Glayderson Jonathan Nunes, and Lidiany C. Santos

Um Estudo Sistemático sobre Abordagens para lidar com a Evolução e Customização de Sistemas

Alternative Title: A Systematic Study on Approaches to deal with the Systems' Evolution and Customization

Fernanda A. Passos
Universidade Federal de Sergipe
Marechal Rondon, Rosa Elze
São Cristóvão – SE, Brasil
fernanda@ufs.br

Kleber T. O. Santos
Universidade Federal de Sergipe
Marechal Rondon, Rosa Elze
São Cristóvão – SE, Brasil
klebertarcisio@yahoo.com.br

Raphael F. S. Barreto
Universidade Federal de Sergipe
Marechal Rondon, Rosa Elze
São Cristóvão – SE, Brasil
rfsbarreto@gmail.com

Galileu S. de Jesus
Universidade Federal de Sergipe
Marechal Rondon, Rosa Elze
São Cristóvão – SE, Brasil
galilasmb@gmail.com

Glayderson J. Nunes
Universidade Federal de Sergipe
Marechal Rondon, Rosa Elze
São Cristóvão – SE, Brasil
glaydersonjonathan@gmail.com

Lidiany C. Santos
Universidade Federal de Sergipe
Marechal Rondon, Rosa Elze
São Cristóvão – SE, Brasil
lidianycs@gmail.com

RESUMO

Desenvolvedores de sistemas muitas vezes enfrentam problemas na manutenção e evolução de sistemas de software quando precisam customizar produtos para atender às necessidades de diferentes clientes, através da criação de novos componentes e modificação do código fonte existente. Neste trabalho, é apresentada uma análise comparativa das abordagens que lidam com variações em Linhas de Produto de Software (LPS) através de um estudo rigoroso do estado da arte, observando sua aplicabilidade para lidar com customizações.

Palavras-Chave

Customização de Software; Variabilidade; Estudo Sistemático; Análise Comparativa.

ABSTRACT

System developers often face problems in the maintenance and evolution of software systems when they need to customize products to meet different customers needs, by creating new components and modifying existing source code. In this work, it is presented a comparative analysis of existing approaches that deal with variations in Software Product Lines (LPS) through a rigorous study of the state of the art, observing their applicability to handle customizations.

Categories and Subject Descriptors

K.6.3 [**Software Management**]: Software maintenance

General Terms

Management, Languages.

Keywords

Software Customization; Variability; Systematic Study; Comparative Analysis.

1. INTRODUÇÃO

O processo de adaptação e manutenção de sistemas adquiridos de outras empresas desenvolvedoras desses sistemas requer atividades inevitáveis em aplicativos de software desde lidar com mudanças no domínio de aplicação a requisitos dos usuários referentes às melhorias na aplicação, além de, novos requisitos. Desta forma, um único aplicativo de software nem sempre consegue satisfazer aos diferentes requisitos de grupos de usuários distintos. Este cenário é frequente e ocorre em sistemas customizados pela equipe de organizações adquirentes, como pode ser observado no SIG (Sistema Integrado de Gestão), descrito em mais detalhes no artigo [24]. Essse sistema foi desenvolvido pela Universidade Federal do Rio Grande do Norte (UFRN), adquirido e adaptado pela Universidade Federal de Sergipe (UFS).

As atividades envolvidas neste processo não são triviais, porque as variações nos requisitos para satisfazer os usuários finais são relativamente frequentes e muitas vezes afetam vários pontos do sistema original adquirido. Sendo assim, torna-se difícil o gerenciamento das variações de forma

modular. Além disso, têm que lidar com as evoluções do sistema original realizadas pelos criadores que normalmente geram impacto sobre as variações desenvolvidas pelos adquirentes, por estas serem realizadas diretamente sobre o código base do sistema original, conforme discutido no trabalho [24]. Consequentemente, o processo de adaptação e manutenção precisa ser aprimorado, necessitando de abordagens que possibilitem:

- A criação de produtos customizáveis perante as necessidades específicas de cada organização;
- A modularização das implementações das variações no sistema sem que seja preciso modificar diretamente o código base do sistema original;
- Gerenciamento das variações realizadas pelas organizações adquirentes no sistema original através da modularização das mesmas;
- Agilidade no processo de sincronização de versões ao ser disponibilizada uma nova versão pelo fornecedor do sistema original;

Diante deste cenário, a utilização de técnicas que lidam com variações em Linhas de Produtos de Software [10], através de seus princípios de variabilidade, podem ser capazes de proporcionar a modularização e o gerenciamento das variações, aprimorando o processo de adaptação e manutenção em sistemas customizados e a evolução destes em sincronia com as novas versões disponibilizadas pelo fornecedor. Em trabalhos anteriores [24] e [26] foi analisada a aplicabilidade de três técnicas para alguns tipos de variações levantadas no cenário atual de customização do SIG, sendo elas: AspectJ [17], o Java Transformation System (JaTS) [9] e a XML-based Variant Configuration Language (XVCL) [16]. Estas técnicas, se mostraram boas candidatas à resolução do problema.

Este artigo traz como contribuição a realização de um estudo mais rigoroso do estado da arte sobre abordagens existentes para lidar com customizações no código base de sistemas adquiridos de outras empresas. Neste estudo, foi observada a aplicabilidade de cada abordagem seguindo alguns critérios relevantes adotados para classificação das técnicas. De posse desta classificação, foi realizado um comparativo das abordagens selecionadas no estudo, indicando possíveis técnicas a serem utilizadas no processo de adaptação e manutenção desses sistemas adquiridos.

É importante ressaltar que as abordagens levantadas através desse estudo para serem utilizadas não consistem na refatoração do código base do sistema adquirido, mas na adaptação do sistema sem modificar diretamente o código base do sistema original. A refatoração, apesar de facilitar a customização de uma versão específica, dificultaria bastante a atualização para versões subsequentes do sistema. Isto acontece porque o código estaria muito diferente da versão original, necessitando-se analisar a versão anterior, a nova e a customizada ao mesmo tempo. Desta maneira, a análise do que evoluiu de uma versão para a outra torna-se complexa.

As seções seguintes do artigo estão organizadas da seguinte forma. A Seção 2 descreve como foi realizado o estudo sistemático sobre o atual estado da arte referente às abordagens que lidam com as customizações em sistemas. Na Seção 3 são apresentadas as técnicas selecionadas no estudo, classificando-as de acordo com as diferentes abordagens de implementação de LPS e segundo critérios relevantes à implementação de variabilidades em sistemas customizáveis. A Seção 4 apresenta uma análise comparativa entre as técnicas conforme suas abordagens de implementação, indicando dentre elas as que são consideradas boas candidatas para a resolução do problema de pesquisa. Logo em seguida, na seção 5 é descrita em mais detalhes as técnicas escolhidas através de suas características e seu funcionamento. Finalmente, a Seção 6 traz as conclusões e trabalhos futuros.

2. ESTUDO SISTEMÁTICO

O processo de revisão sistemática adotado neste trabalho foi o proposto por Kitchenham [18].

A pesquisa tem como questão central: Quais técnicas vêm sendo utilizadas para lidar com customizações (variações) no código a fim de melhorar o processo de adaptação e manutenção em sistemas passíveis de customização?

Os critérios de inclusão adotados foram: (a) Nenhuma restrição no período de publicação; (b) Os trabalhos devem apresentar textos integrais de estudos, escritos em Português ou Inglês, em um formato eletrônico disponível na WEB; (c) Os estudos encontrados devem apresentar técnicas ou ferramentas que lidam com variabilidades em sistemas. Para os de exclusão: estudos que não apresentam pelo menos uma técnica ou ferramenta que lide com variabilidades em sistemas.

O estudo foi realizado por meio de busca eletrônica: (a) No portal da CAPES[1] por periódicos escritos na língua inglesa, referenciando as principais bases de pesquisa como: IEEE Xplore Digital Library, ScienceDirect, Springer Link e ACM Digital Library. Além, das bases de teses e dissertações relacionadas com o tema de pesquisa na língua portuguesa. (b) No Google Scholar[2] por trabalhos indexados por algumas das bases de pesquisa, escritos na língua inglesa e portuguesa.

As strings de busca utilizadas em inglês: (techniques or tool) and "variability in code"; (techniques or tool) and "variability managing"; (techniques or tool) and "variability customization"; (techniques or tool) and "software variability"; (techniques or tool) and "variability implementation". Em português foram: "transformação de código"; "variabilidades em código"; "gerenciamento de variabilidades"; "customização de variabilidades"; "implementação de variabilidades".

Para a fonte de busca Google Scholar (Inglês) e CAPES (Teses e Dissertações), foi necessário realizar modificações em algumas expressões, para em alguns casos ampliar a abrangência da busca e em outros delimitar mais o escopo da busca devido aos resultados serem bastante abrangentes.

A condução foi realizada em um mês. A partir das strings de buscas foram retornados um conjunto de 1481 registros, que após análise inicialmente do título e do resumo e em alguns casos da introdução resultou na seleção de 31 trabalhos, nestes não foram encontrados trabalhos repetidos. O resultado final do estudo está apresentada na Tabela 1, em que na coluna **Res. Enc.** refere-se ao quantitativo de Resultados Encontrados e a coluna **Trab. Sel.** ao quantitativo de Trabalhos Selecionados. Vale ressaltar que muitos dos trabalhos selecionados não tratam especificamente de uma técnica, mas citam ou fazem comparações entre elas. Porém, através das referências às técnicas foi possível encon-

[1] `http://www.periodicos.capes.gov.br`
[2] `http://scholar.google.com.br`

trar o periódico e ainda obter referências de outras técnicas na seção de Trabalhos Relacionados destes periódicos.

Tabela 1: Resultados da Revisão Sistemática, 2013.

Base	Res. Enc.	Trab. Sel.
ACM Digital Library	37	2
IEEExplore	159	6
ScienceDirect	135	0
Google Scholar (Inglês)	837	18
Google Scholar (Português)	141	3
CAPES (Teses e Dissertações)	172	2
Total	1481	31

Na próxima seção serão apresentadas as técnicas levantadas neste estudo classificando-as em abordagens de implementação de variações em LPS, segundo as seguintes categorias: anotativa, composicional, modelagem e transformacional.

3. ABORDAGENS QUE LIDAM COM VARIAÇÃO EM LPS

A implementação de variabilidades em Linhas de Produtos de Software (LPS) permite construir *assets* customizáveis e extensíveis para a concepção de diferentes produtos de software derivados de uma LPS. Existem diferentes abordagens para a implementação das variantes em LPS, podendo ser classificadas nas seguintes categorias: anotativa, composicional, modelagem e transformacional.

As abordagens são apresentadas nas subseções seguintes e nelas as técnicas para LPS encontradas no estudo sistemático relatado na Seção 2 são enquadradas e classificadas seguindo critérios adicionais relevantes à implementação de variabilidades em sistemas. Os critérios foram definidos para verificar dentre as técnicas levantadas, quais se enquadrariam em uma abordagem para aprimorar o processo de adaptação e manutenção de sistemas customizados. Para isso, a técnica deve permitir o suporte à fase de implementação do ciclo de vida do software, à derivação de produtos, à criação de artefatos de software customizáveis com suporte no nível de granularidade grossa e fina e à separação das variações do código base do sistema original.

Os critérios escolhidos para classificação das técnicas foram: **(i) Configuração do Produto**: especificação de um produto da LPS a ser gerado pelo processo de derivação de produtos baseado na configuração de *features*; **(ii) Fases do ciclo de Vida da LPS**: as fases do ciclo de vida do software suportadas pela ferramenta no desenvolvimento de LPS; **(iii) Seleção dos Componentes Concretos**: *assets* de código concretos (classes, aspectos, arquivos e outros) mapeados às *features* selecionadas para geração do produto final; **(iv) Geração dos Componentes Flexíveis**: suporta adaptações, composições e transformações dos *assets* reusáveis para contemplar possíveis variações conforme as *features* selecionadas para geração do produto final; **(v) Granularidade da Variabilidade**: o nível de granularidade (grossa ou fina) suportada pela técnica para implementação dos *assets* reusáveis e de variações nos mesmos. Nível de granularidade grossa possibilita adicionar novas classes, aspectos, métodos e atributos ou estender pontos de variações explícitos. E o nível de granularidade fina permite introduzir trechos de código a métodos existentes, estender expressões ou assinaturas de métodos; **(vi) Separação da Variação do Código**: implementação das variações separada do código base do sistema original.

As classificações e dados informados nas tabelas apresentadas nas subseções seguintes foram obtidos da análise de informações disponíveis em documentações e periódicos publicados sobre cada técnica. E as seguintes abreviações são utilizadas: (i) Fases do Ciclo de Vida: C (Concepção), A (Análise), P (Projeto), I (Implementação) e TD (Todas); (ii) Granularidade da Variabilidade: G (Grossa) e F (Fina).

3.1 Abordagem Anotativa

Na abordagem anotativa, os fragmentos de código de *features* (características) são anotados em uma base de código comum e podem ser usados não somente para configurar variantes estaticamente antes da compilação, mas também para habilitar ou desabilitar *features* em tempo de execução. As anotações também podem ser realizadas em outros artefatos como em modelos [25]. É utilizado nesta abordagem o pré-processador, um programa executado antes do compilador para analisar o código fonte a ser compilado e, de acordo com as anotações adicionadas ao mesmo, são executadas alterações antes de repassá-lo ao compilador. Dentre as técnicas levantadas, foram identificadas 3 propostas de abordagem anotativa: *Antenna*[3], *XVCL* (*XML-based Variant Configuration Language*) [16] e *CIDE* (*Colored Integrated Development Environment*)[4].

Tabela 2: Classificando as Técnicas de LPS de Abordagem Anotativa.

Técnica	Config. do Prod.	Fases do Ciclo de Vida	Sel. dos Comp. Concretos	Ger. dos Comp. Flex.	Granu. da Var.	Sep. da Var. do Código
Antenna	SIM	I	NÃO	SIM	SIM(G,F)	NÃO
CIDE	SIM	I	SIM	SIM	SIM(G,F)	NÃO
XVCL	SIM	I	NÃO	SIM	SIM(G,F)	SIM

De acordo com a Tabela 2 as técnicas desta abordagem porporcionam a configuração de produtos e a geração de componentes flexíveis através do mecanismo de anotação explícita com granularidade fina diretamente no código fonte podendo ser textual ou visual. As técnicas Antenna e XVCL utilizam anotações textuais por meio de diretivas de préprocessamento ou de comandos específicos da técnica diretamente no código fonte. Diferentemente, a CIDE baseia-se em anotações visuais através da marcação do código por uma cor de fundo, o que proporciona uma separação virtual através das cores das *features*, mas não há uma separação de fato do código base do sistema. No entanto, a técnica XVCL possui uma vantagem em relação a Antenna e CIDE que é a separação do trecho de código que contempla a customização do código base do sistema original em outro arquivo, requerendo apenas a anotação dos pontos no código base do sistema original onde estes trechos de código irão ser introduzidos.

3.2 Abordagem Composicional

[3] `http://antenna.sourceforge.net/wtkpreprocess.php`
[4] `http://wwwiti.cs.uni-magdeburg.de/iti_db/research/cide/`

Na abordagem composicional, as *features* são implementadas separadamente em módulos distintos (arquivos, classes, pacotes, plug-ins e outros) e são geradas instâncias da linha de produtos através da composição destes módulos. Para gerar variantes, estes módulos podem ser compostos em diferentes combinações. Os mecanismos de implementação utilizados são: componentes, *mixin layers*, classes virtuais, aspectos e outros [25]. Nesta abordagem foram identificadas 7 técnicas: *AHEAD* [3], *FeatureHouse*[5], *JPEL* [6], *AspectJ* [17], *CaesarJ* [7], *JBOSS AOP*[8] e Spring AOP[9]. Estas últimas quatro técnicas pertencem ao paradigma de Programação Orientado a Aspectos (POA), tendo como unidade modular o aspecto.

As técnicas enquadradas nesta abordagem, como mostra a classificação realizada na Tabela 3, possibilitam a configuração de produtos e a geração de componentes flexíveis através da implementação de módulos distintos com granularidade que vai de grossa à fina contemplando as customizações necessárias ao sistema original mediante a composição destes módulos ao código base do sistema. As técnicas AHEAD, FeatureHouse e POA possibilitam a adição de novos campos e métodos a classes do sistema afetando a estrutura estática, ou ainda a adição de comportamento extra no início ou final dos métodos existentes. Diferentemente do JPEL que é limitado a implementar variações de valor, através de arquivos de parametrização para alterar os valores das variáveis locais, atributos, constantes usadas para calcular uma expressão no sistema e a testes condicionais referentes à adição ou não de variabilidades comportamentais implementadas diretamente no código base do sistema original, não sendo concebível alterar a estrutura do programa.

Tabela 3: Classificando as Técnicas de LPS de Abordagem Composicional

Técnica	Conf. do Prod.	Fases do Ciclo de Vida	Sel. dos Comp. Concretos	Ger. dos Comp. Flex.	Granu. da Var.	Sep. da Var. do Código
AHEAD	SIM	I	NÃO	SIM	SIM(G,F)	SIM
FeatureHouse	SIM	I	NÃO	SIM	SIM(G,F)	SIM
JPEL	SIM	I	NÃO	SIM	SIM(F)	NÃO
POA	SIM	I	NÃO	SIM	SIM(G,F)	SIM

3.3 Abordagem de Modelagem de LPS

Na abordagem de modelagem de LPS é possível especificar as *features* que contemplam o escopo de toda a linha de produtos, as dependências, os relacionamentos e as restrições entre as mesmas. E ainda sistematizar o processo de derivação de produtos que envolvem a seleção, composição e customização dos artefatos de código que implementam uma LPS com o objetivo de atender a configuração de *features*.

Foram levantadas nesta abordagem 21 propostas: *CodeScoping [21], CONSUL [7], COVAMOF [27], DOPLER*[10], *FAMA FW [6], FeatureIDE [29], FeatureMapper [15], FeaturePlugin [1], Gears [20], GenArch*[11], *Holmes [28], Kumbang [19], Ménage [12], PLUSEE [14], pure::variants*[12], *Requiline [30], S2T2 [8], S.P.L.O.T. [22]* , *VarMod*[13], *V-Manage* [5] e *XToF* [13].

Segundo a Tabela 4, algumas técnicas adotam somente uma abordagem de modelagem de LPS para a especificação das *features* que compõem o escopo de todo o sistema modelando as partes comuns e variáveis entre os diversos produtos de software que formam uma LPS. Algumas destas técnicas, como FeatureMapper, FeaturePlugin, Ménage e Kumbang, somente realizam a modelagem, isto é, define os relacionamentos e as dependências entre as *features* cobrindo somente a fase de projeto do ciclo de vida da LPS. E outras como: COVAMOF, DOPLER, Holmes e CodeScoping realizam também o processo efetivo de construção do produto baseado na seleção de componentes concretos através do mapeamento dos *assets* reusáveis às *features* selecionadas para compor o produto final. Porém, essas técnicas não permitem realizar customizações nos *assets* que possibilitem a existência de diferentes especificações de código (variações) para uma mesma *feature*.

Por outro lado, as técnicas pure::variants, Gears, XToF, GenArch e CONSUL, além de realizarem a especificação das *features* correspondentes ao domínio da linha de produto e a configuração do produto (através de um subconjunto válido de *features* que satisfazem as restrições presentes no modelo de *features* especificado para obtenção de um produto específico da LPS), possuem como recurso adicional a geração de componentes flexíveis para efetuar customizações nos *assets* com granularidade grossa através de transformações e da composição de novos *assets* para possíveis adaptações às *features* e a evolução da LPS devido a novos requisitos de domínio.

3.4 Abordagem Transformacional

Na abordagem transformacional, os sistemas de transformação de programa fornecem mecanismos para a realização de modificações arbitrárias e não apenas a composição de módulos, como transformar a linguagem fonte de um programa num novo programa com linguagem diferente (por exemplo, na compilação) ou transformar um programa em um novo programa na mesma linguagem (por exemplo, na refatoração). Os mecanismos de implementação encontrados foram: Linguagem Específica de Domínio (Domain Specific Language - DSL[14]) e meta-programação para definir regras de transformações de programas [25]. Em relação a este tipo de abordagem foram encontradas 5 propostas: DMS (*Design Maintenance System*) [4], JaTS (*Java Transformation System*)[15], *MetaJ*[16], *Stratego/XT*[17] e *TXL*(*Turing eXtender Language*) [18].

[5] http://www.fosd.de/fh
[6] http://sourceforge.net/projects/jpel/
[7] http://www.caesarj.org/
[8] http://docs.jboss.org/jbossaop/docs/index.html
[9] http://static.springsource.org/spring/docs/2.5.5/reference/aop.html
[10] http://ase.jku.at/dopler/
[11] http://www.inf.puc-rio.br/~ecirilo/genarch/
[12] http://www.pure-systems.com/fileadmin/downloads/pv-whitepaper-en-04.pdf
[13] http://www.sse.uni-essen.de/swpl/SEGOS-VM-Tool/
[14] A Domain Specific Language (DSL) é uma linguagem textual ou gráfica que proporciona abstrações de primeira classe representando diretamente os conceitos de domínio da aplicação.
[15] www.cin.ufpe.br/~jats/
[16] www.emn.fr/sudholt/research/metaj/
[17] http://www.strategoxt.org
[18] www.txl.ca

Tabela 4: Classificando as Técnicas de LPS de Abordagem de Modelagem

Técnica	Conf. do Prod.	Fases do Ciclo de Vida	Sel. dos Comp. Concretos	Ger. dos Comp. Flex.	Granu. da Var.	Sep. da Var. do Código
CodeScoping	NÃO	P	SIM	NÃO	NÃO	NÃO
CONSUL	SIM	A,P,I	SIM	SIM	SIM(G)	SIM
COVAMOF	SIM	A,P,I	SIM	NÃO	NÃO	NÃO
DOPLER	SIM	C,A,P,I	SIM	NÃO	NÃO	NÃO
FAMA-FW	NÃO	A, P	NÃO	NÃO	NÃO	NÃO
FeatureIDE	SIM	A,P,I	SIM	SIM	SIM(G)	SIM
FeatureMapper	SIM	P	NÃO	NÃO	NÃO	NÃO
FeaturePlugin	SIM	P	NÃO	NÃO	NÃO	NÃO
Gears	SIM	A,P,I	SIM	SIM	SIM(G)	SIM
GenArch	SIM	P, I	SIM	SIM	SIM(G)	SIM
Holmes	NÃO	TD	SIM	-	-	-
Kumbang	SIM	P	NÃO	NÃO	NÃO	NÃO
Ménage	NÃO	P	NÃO	NÃO	NÃO	NÃO
PLUSEE	SIM	A, P	NÃO	NÃO	NÃO	NÃO
pure::variants	SIM	TD	SIM	SIM	SIM(G)	SIM
RequiLine	SIM	C,A,P	NÃO	NÃO	NÃO	NÃO
S2T2	NÃO	A, P	NÃO	NÃO	NÃO	NÃO
S.P.L.O.T.	SIM	A, P	NÃO	NÃO	NÃO	NÃO
VarMod	SIM	A, P	NÃO	NÃO	NÃO	NÃO
V-Manage	SIM	C,A,P,I	SIM	SIM	SIM(G)	SIM
XToF	SIM	A,P,I	NÃO	SIM	SIM(G)	NÃO

Tabela 5: Classificando as Técnicas de LPS de Abordagem Transformacional.

Técnica	Conf. do Prod.	Fases do Ciclo de Vida	Sel. dos Comp. Concretos	Ger. dos Comp. Flex.	Granu. da Var.	Sep. da Var. do Código
DMS	SIM	I	NÃO	SIM	SIM(G,F)	SIM
JaTS	SIM	I	NÃO	SIM	SIM(G,F)	SIM
MetaJ	SIM	I	NÃO	SIM	SIM(G,F)	SIM
Stratego/XT	SIM	I	NÃO	SIM	SIM(G,F)	SIM
TXL	SIM	I	NÃO	SIM	SIM(G,F)	SIM

Essas técnicas utilizam o casamento de padrão a ser correspondido, tornando possível a configuração de produtos e a geração de componentes flexíveis para efetuar customizações nos *assets* separadamente do código base do sistema original proporcionando um nível de granularidade que vai de grossa a fina, como mostra a classificação realizada na Tabela 5. As técnicas *Stratego/XT, DMS* e *TXL* são baseadas em regras de reescrita para realizar as transformações na linguagem alvo de acordo com o padrão informado. A MetaJ e o JaTS utilizam *templates* que definem um padrão de código cujas sintaxes são bastante similares. Sendo que na MetaJ, os *templates* são então traduzidos para uma classe Java e o casamento é verificado através do método *match()* em que o código fonte é passado como parâmetro. O comportamento ao encontrar casamentos de padrões deve ser programado pelo desenvolvedor na linguagem Java. Já o JaTS utiliza *templates* de casamento e de geração. Os *templates* de casamento são descritos com base nos padrões do programa que se deseja modificar, enquanto que os de geração definem as modificações que serão produzidas pela transformação do programa.

4. ANÁLISE COMPARATIVA DAS TÉCNICAS DE LPS

As técnicas apresentadas até então, foram classificadas na seção anterior de acordo com as diferentes abordagens de implementação de LPS e segundo critérios adicionais relevantes à implementação de variabilidades em sistemas. Esta seção traz uma análise comparativa das técnicas LPS em suas abordagens de implementação, tendo como contribuição a análise mais focada na indicação de algumas das técnicas mencionadas na Seção 3, por estas serem consideradas boas candidatas para a resolução do problema de pesquisa.

Na análise realizada sobre a Seção 3.3 referente às técnicas de abordagem de modelagem de LPS, foi verificado que apesar de existirem técnicas que efetuam customizações nos *assets* com granularidade grossa através de mecanismos de implementação como, anotação e composição, as mesmas necessitam das especificações das *features* que compõem o escopo do sistema para que seja possível gerar diferentes configurações de produtos através da seleção, transformação ou combinação dos *assets* associados às *features* selecionadas no processo de derivação do produto. Sendo assim, os mecanismos de implementação de variabilidade em LPS proposto por essa abordagem não se alinham com os objetivos deste trabalho, que refere-se à customização do código fonte de sistemas originais desenvolvidos por empresas fornecedoras de software e adquiridos por outras organizações sem a realização de refatorações no código original cobrindo somente a fase de implementação e não contemplando a especificação e a modelagem de *features* do domínio, as quais seriam necessárias para adoção de uma das técnicas de abordagem de modelagem de LPS.

As técnicas da Seção 3.1 de abordagem anotativa utilizam do mecanismo de anotação explícita com granularidade fina diretamente no código fonte o que de certa forma inviabilizaria a sua utilização nesta pesquisa, pois os problemas de manutenção e evolução do sistema continuariam devido à necessidade de reintroduzir as customizações efetuadas ao sistema original a cada nova versão disponibilizada pelo fornecedor. No entanto, a técnica XVCL foi uma das técnicas escolhidas por proporcionar algumas vantagens: a delimitação exata do trecho de código em que ocorre a alteração; a separação do trecho de código que contempla a customização do código base do sistema original em outro arquivo requerendo apenas a anotação dos pontos no código base do sistema original em que estes trechos de código irão ser introduzidos; e a facilidade de aprendizado da técnica, pois não se faz necessário que o desenvolvedor possua um amplo conhecimento da técnica para entender seu funcionamento e seus comandos.

Diferentemente das técnicas anotativas, nas composicionais (Seção 3.2) as variações podem ser implementadas sem anotações explícitas no código original através da separação das variabilidades existentes das *features* do código base do sistema em módulos distintos com granularidade que vai de grossa a fina, promovendo assim a modularização dos interesses (funcionalidades). A POA (através do AspectJ) e a FeatureHouse foram as escolhidas nesta abordagem por proporcionar modificação na estrutura estática de um sistema adicionando atributos, métodos e construtores a uma classe, ou ainda a adição de comportamento extra no início ou final dos métodos existentes. Com a POA é possível ainda afetar a estrutura dinâmica de um programa mediante a in-

terceptação de pontos no fluxo de execução do programa, chamados *join points*, definidos em pointcuts e de um conjunto de *advice* que podem executar comportamento antes e/ou depois de *join points*, e até substituir um comportamento existente no programa por outro. O AHEAD não foi selecionado devido à necessidade de transferência completa do código do sistema para arquivos *".jak"*, pois para utilização desta técnica o desenvolvedor deve separar o código em arquivos (*".jak"*) diferentes de acordo com as features. Para a geração de instâncias da LPS deve-se escolher quais das features serão incluídas, então a ferramenta de composição do AHEAD faz a junção dos arquivos de cada feature em um novo arquivo *".jak"*. E logo após essa junção o arquivo é transformado em um arquivo *Java*.

As abordagens somente transformacionais da Seção 3.4, como os sistemas de transformação de programa Stratego/XT, DMS e TXL são técnicas que possuem um nível de complexidade alto na implementação, visto que realizam modificações arbitrárias, normalmente incluindo a remoção ou reestruturação completa da sintaxe concreta da linguagem objeto. Portanto, não são frequentemente adotados na prática para implementar variações em LPS. A MetaJ é uma técnica de transformação semelhante ao JaTS sendo que a linguagem de *templates* é muito simples e não suporta transformações mais complexas sendo necessário a definição de métodos, por outro lado as transformações descritas de forma declarativa, como em JaTS, são de certa forma mais simples de serem escritas, devido à linguagem de *templates* ser próxima da template da linguagem objeto. No entanto, o JaTS apesar de ser um bom candidato não tem acompanhado as atualizações da linguagem Java, gerando assim uma incompatibilidade ao usar as construções novas da linguagem Java, como *Generics*.

Deste modo, as técnicas indicadas de acordo com os critérios relevantes, mencionados neste artigo, à implementação de variabilidades em sistemas são: as técnicas de abordagem composicional, POA (através do AspectJ) e a FeatureHouse, e a técnica de abordagem anotativa XVCL. As principais razões pela escolha dessas técnicas foram o suporte à implementação de variações em diferentes níveis de granularidade e a separação das mesmas do código sem que seja necessário alterar diretamente o código base do sistema original, refletindo-se em uma vantagem durante a evolução do sistema. A XVCL, apesar de não suportar a separação total das variações, porque exige a marcação dos pontos aonde serão introduzidas as variações (estas sim estão separadas em arquivos), foi selecionada principalmente pela simplicidade de aprendizado e utilização.

5. TÉCNICAS DE LPS ESCOLHIDAS

Nesta seção, serão apresentadas as técnicas AspectJ, FeatureHouse e XVCL escolhidas para a implementação das variantes em sistemas customizáveis descrevendo as suas principais características e seu funcionamento.

AspectJ [17] é uma extensão da linguagem Java para suportar o paradigma de Programação Orientada a Aspectos. Esse paradigma tem como objetivo o suporte à modularização dos interesses transversais por meio de abstrações que possibilitem a separação e composição destes interesses no processo de desenvolvimento de software.

O princípio da separação de interesses (*concerns*) foi introduzido por Dijkstra[11], tendo como objetivo dividir o domínio do sistema em partes menores, ou seja, separados em módulos. O termo interesse é utilizado no contexto da Engenharia de Software para representar requisitos que precisam estar contidos nos sistemas. Os interesses podem ser vistos como requisitos funcionais, tais como regras de negócios, ou não funcionais, como gerenciamento de transação e persistência. Os interesses transversais (*crosscutting concerns*) são portanto interesses que afetam vários módulos do sistema e são difíceis de separar por estarem espalhados por todo o sistema e entrelaçados com os outros interesses dentro de um mesmo módulo. Exemplos destes interesses são dentre outros: *logging*, *tracing*, distribuição e autorização.

Com a utilização desse paradigma é introduzida uma nova abstração, chamada aspecto, em que os interesses transversais são encapsulados e os mecanismos de integração da POA realizam a composição do aspecto aos artefatos de software (componentes de software da programação orientada a objetos tais como classes, métodos e atributos) num processo chamado *weaving*.

Sendo assim, uma aplicação AspectJ é composta por classes e aspectos. Os aspectos permitem modificar a estrutura estática de um sistema, por exemplo, adicionando atributos, métodos e construtores a uma classe e alterando a hierarquia do sistema. Além disso, é possível afetar a estrutura dinâmica de um programa através da interceptação de pontos no fluxo de execução, chamados *join points* e da adição de comportamento definido em um *advice*.

A FeatureHouse [19] é uma arquitetura geral de composição de software baseada em um modelo independente de linguagem de artefatos de software e um mecanismo de plug-in automático para a integração de novos artefatos de linguagens sendo apoiada por um framework e um conjunto de ferramentas.

A técnica generaliza e integra os artefatos de software através da ferramenta FSTCOMPOSER que se baseia em um modelo geral da estrutura de artefatos de software, chamado de modelo de Árvore da Estrutura de *Features* (*Feature Structure Tree* - FST). Uma FST representa a estrutura essencial de um artefato de software e resumos a partir de detalhes específicos da linguagem. Por exemplo, um artefato escrito em Java contém pacotes, classes, métodos e assim por diante, que são representados por nós em FST.

A composição dos artefatos de software é realizada através do princípio de sobreposição, pois duas FSTs são sobrepostos pelo *merge* dos nós, identificados pelos nomes, tipos e posições relativas, começando a partir da raiz e descendo recursivamente. No entanto, a composição de duas folhas de uma FST que contenham conteúdo a mais, por exemplo, dois corpos de métodos exigem um tratamento especial. A razão é que o conteúdo dos métodos não é representado como uma sub-árvore, mas como um simples texto. Nesta situação é realizado o merge dos corpos dos métodos através de *override*, em que a inclusão da palavra reservada `original` define como os corpos dos métodos serão mesclados, que de certa forma se assemelha ao `super` do Java. A palavra reservada `original` é uma meta-notação que desaparece após a composição. O interpretador Java trata `original` como uma chamada de método. Durante a composição de dois corpos de métodos, a palavra reservada `original` será procurada e substituída pelo corpo do método original. As Figuras 1 ilustra o processo de sobreposição de FSTs.

A técnica XVCL [16] é baseada numa abordagem de ano-

[19] `http://www.fosd.de/fh`

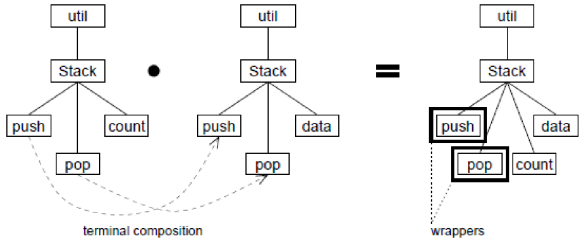

Figura 1: Composição de FSTs [2].

tação permitindo realizar marcações semelhante ao XML (*eXtensible Markup Language*) em trechos específicos do código indepentende de linguagem, sendo incentivada para os casos de reuso e de adaptação do software com a finalidade de configurar variantes em uma variedade de ativos de software, tais como modelo, o código de programa de domínio, casos de teste e outros. É geralmente utilizada no desenvolvimento de Linhas de Produtos de Software (LPS), visto que uma família de software necessita constantemente adaptar o sistema aos requisitos exigidos pelo cliente.

A técnica é embasada no conceito de programação de *frame* em que cada definição de *frame* é chamada de *X-FRAME*. Cada *X-FRAME* é um componente adaptável programado em uma estrutura de arquivo XML. Neste arquivo está presente um trecho de código fonte com marcações de comandos XVCL, estes comandos possibilitam que trechos do código fonte possam ter diversos comportamentos diferentes de acordo com os requisitos do cliente, demonstrando a capacidade de adaptação fornecida pela técnica.

Um *X-FRAME* pode invocar outros *X-FRAMES* através de um processo chamado de adaptação, que permite realizar a customização do programa. Cada *frame* adapta qualquer um ou todos seus descendentes e cada *frame* pode ser adaptado pelo *frame* pai, sendo então organizados em hierarquia como demonstra a Figura 2. O conceito mais importante na adaptação é o ponto de interrupção que marca pontos no código fonte onde será feita inserção de trecho de código.

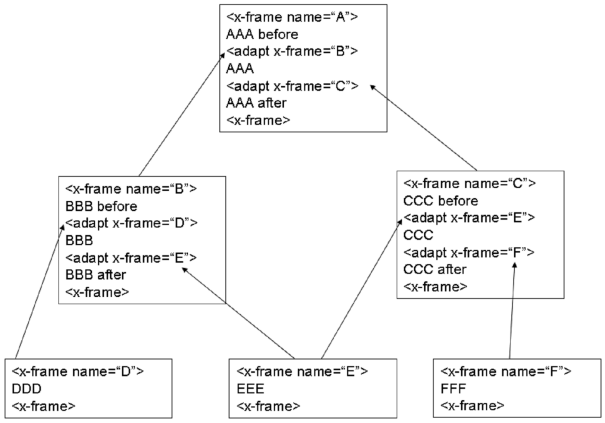

Figura 2: Funcionamento do XVCL [16].

Na Figura 2, todos os *X-FRAMES* estão conectados ao *X-FRAME* raiz A, que é o ponto de partida para gerar o programa com as adaptações a partir das anotações realizadas. Já os *X-FRAMES* filhos contêm o código fonte com as adaptações que precisam ser realizadas nos pontos de interrupção. Portanto, cada programa XVCL consiste de um *X-FRAME* raiz, sendo referenciado como um arquivo de especificação, e da composição de vários *X-FRAMES*. O processador XVCL analisa o arquivo de especificação e constrói o programa, realizando a composição com todos os outros *X-FRAMES* utilizados.

6. CONCLUSÕES

Este artigo traz um estudo sistemático sobre técnicas que podem auxiliar na implementação de customizações (variações) no código base de sistemas adquiridos de empresas desenvolvedoras de aplicativos de software. Após o levantamento do estudo, foi realizada a classificação das técnicas encontradas de acordo com as abordagens de implementação de LPS e os critérios adicionais importantes à customização de sistemas adquiridos. Em seguida, foi feita a comparação dessas abordagens, obtendo a indicação de AspectJ, FeatureHouse e XVCL como as técnicas mais apropriadas ao problema discutido neste artigo. Diante da contribuição deste trabalho, as técnicas indicadas foram avaliadas num contexto real no trabalho [23], através do planejamento e execução de um quasi-experimento.

Como trabalho futuro, está em processo de realização uma análise mais rigorosa dos resultados do quasi-experimento do ponto de vista estatístico. Esta análise, visa observar estatisticamente o desempenho e a confiança das técnicas indicadas neste artigo no processo de adaptação e manutenção de sistemas customizados.

Agradecimentos

Gostaríamos de agradecer ao CNPq, CAPES e Fapitec, agências brasileiras de fomento à pesquisa, por apoiar parcialmente este trabalho.

7. ADDITIONAL AUTHORS

Alberto Costa Neto (Universidade Federal de Sergipe, email: `alberto@ufs.br`).

8. REFERÊNCIAS

[1] M. Antkiewicz and K. Czarnecki. Featureplugin: feature modeling plug-in for eclipse. In *Proceedings of the 2004 OOPSLA workshop on eclipse technology eXchange*, pages 67–72, Vancouver, British Columbia, Canada, 2004. ACM.

[2] S. Apel and C. Lengauer. Superimposition: A language-independent approach to software composition. In *7th International Symposium Software Composition*, pages 20–35, Budapest, Hungary, 2008. Springer.

[3] D. Batory. Feature-oriented programming and the ahead tool suite. In *Proceedings of the 26th International Conference on Software Engineering (ICSE '04)*, pages 702–703, Scotland, UK, 2004. IEEE Computer Society.

[4] I. D. Baxter, C. Pidgeon, and M. Mehlich. Dms®: Program transformations for practical scalable software evolution. In *Proceedings of the 26th International Conference on Software Engineering (ICSE '04)*, pages 625–634, Scotland, UK, 2004. IEEE Computer Society.

[5] J. Bayer, S. Gerard, Ø. Haugen, J. Mansell, B. Møller-Pedersen, J. Oldevik, P. Tessier, J.-P. Thibault, and T. Widen. Consolidated product line variability modeling. *Software Product Lines*, pages 195–241, 2006.

[6] D. Benavides, S. Segura, P. Trinidad, and A. Ruiz-Cortés. Fama: Tooling a framework for the automated analysis of feature models. In *Proceeding of the First International Workshop on Variability Modelling of Software-intensive Systems (VaMoS 2007)*, pages 129–134, Limerick, Ireland, 2007. Citeseer.

[7] D. Beuche, H. Papajewski, and W. Schröder-Preikschat. Variability management with feature models. *Science of Computer Programming*, 53(3):333–352, December 2004.

[8] G. Botterweck, M. Janota, and D. Schneeweiss. A design of a configurable feature model configurator. In *3rd International Workshop on Variability Modelling of Software-intensive Systems (VaMoS 2009)*, volume 29, pages 165–168, Sevilla, Spain, 2009. ICB Research Report.

[9] F. Castor, K. Oliveira, A. Souza, G. Santos, and P. Borba. JaTS: A Java Transformation System. In *Proceedinds of XV Brazilian Symposium on Software Engineering (SBES 2001)*, pages 374–379, Rio de Janeiro, Brasil, October 2001.

[10] P. Clements and L. M. Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley, Boston, MA, USA, 2001.

[11] E. W. Dijkstra. *A discipline of programming*, volume 1. Prentice-Hall, Englewood Cliffs, NJ, 1976.

[12] A. Garg, M. Critchlow, P. Chen, C. Van der Westhuizen, and A. Van der Hoek. An environment for managing evolving product line architectures. In *Proceedings of the 19th International Conference on Software Maintenance (ICSM 2003)*, pages 358–367, Amsterdam, The Netherlands, 2003. IEEE.

[13] C. Gauthier, A. Classen, Q. Boucher, P. Heymans, M.-A. Storey, and M. Mendonça. Xtof: A tool for tag-based product line implementation. volume 10, pages 163–166, Linz, Austria, 2010. University of Duisburg-Essen.

[14] H. Gomaa and M. E. Shin. Tool support for software variability management and product derivation in software product lines. In *International Workshop on Software Variability Management for Product Derivation, Software Product Line Conference (SPLC).*, Boston, USA, August 2004.

[15] F. Heidenreich, J. Kopcsek, and C. Wende. Featuremapper: mapping features to models. In *Proceedings of the 30th International Conference on Software Engineering (ICSE)*, pages 943–944, Leipzig, Germany, 2008. ACM.

[16] S. Jarzabek. *XML-based Variant Configuration Language (XVCL), Specification Version 2.10*. Singapore, National University of Singapore edition, Junho 2006.

[17] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. Griswold. Getting Started with AspectJ. *Communications of the ACM*, 44(10):59–65, Oct. 2001.

[18] B. Kitchenham. Procedures for performing systematic reviews. *Joint Technical Report, Computer Science Department*, 33, 2004.

[19] H. Koivu. A tool for modelling software product families – user-centred prototype implementation. Master's thesis, Department of Computer Science and Engineering, Helsinki University of Technology, Espoo, Finland, February 2007.

[20] C. W. Krueger. Software mass customization. *White paper, BigLever Software Inc.*, May 2005.

[21] T. F. L. Medeiros, E. S. Almeida, and S. R. Lemos Meira. Codescoping: A source code based tool to software product lines scoping. In *38th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA'12)*, pages 101–104, Cesme, Izmir, 2012. IEEE.

[22] M. Mendonca, M. Branco, and D. Cowan. S.P.L.O.T.: Software Product Lines online tools. In *Proceedings of the 24th ACM SIGPLAN conference companion on Object Oriented Programming Systems Languages and Applications (OOPSLA'09)*, pages 761–762, Orlando, Florida, USA, October 2009. ACM.

[23] F. Passos, G. Jesus, G. Nunes, R. Barreto, K. Santos, and A. Costa Neto. Experimentando AspectJ como uma abordagem para lidar com a Evolução e Customização de um Sistema Integrado de Gestão. In *WMod2014 - 11th Workshop on Software Modularity* , Maceió, Brasil, 2014.

[24] F. Passos, K. Santos, R. Barreto, and A. Costa Neto. Adaptação e Manutenção de Sistemas Integrados de Gestão apoiados pela Programação Orientada a Aspectos. In *IX Simpósio Brasileiro de Sistemas de Informação*, João Pessoa, Brasil, Maio 2013.

[25] G. Saake, D. Batory, and K. Czarnecki. *Virtual separation of concerns: toward preprocessors 2.0*. PhD thesis, Otto-von-Guericke-Universität Magdeburg, Magdeburg, Germany, May 2010.

[26] K. Santos, R. Barreto, F. Passos, and A. Costa Neto. Utilizando JaTS e XVCL para customização em Sistemas Integrados de Gestão. In *Erbase 2013 - XIII Escola Regional de Computação Bahia Alagoas Sergipe*, Aracaju, Brasil, Abril 2013.

[27] M. Sinnema, S. Deelstra, J. Nijhuis, and J. Bosch. Covamof: A framework for modeling variability in software product families. volume 3154, pages 197–213, Boston, MA, USA, August 2004. Springer Verlag.

[28] G. Succi, J. Yip, and W. Pedrycz. Holmes: an intelligent system to support software product line development. In *Proceedings of the 23rd International Conference on Software Engineering*, pages 829–830, Toronto, Ontario, Canada, May 2001. IEEE Computer Society.

[29] T. Thm, C. Kstner, F. Benduhn, J. Meinicke, G. Saake, and T. Leich. FeatureIDE: An Extensible Framework for Feature-Oriented Software Development. *Science of Computer Programming*, 2012.

[30] T. von der Maßen and H. Lichter. Requiline: A requirements engineering tool for software product lines. In *Proceedings of the 5th Workshop on Product Family Engineering (PFE 2003)*, volume 3014, pages 168–180, Siena, Italy, 2003. Springer.