

PATRÍCIA DE PAULA DIAS DE OLIVEIRA

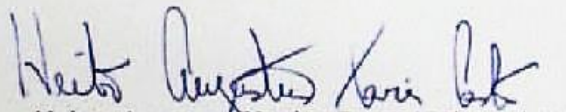
**EXTRAÇÃO DE UMA LINHA DE PRODUTOS DE
SOFTWARE UTILIZANDO COMPILAÇÃO
CONDICIONAL**

Monografia de graduação apresentada ao
Colegiado do Curso de Bacharelado em
Sistemas de Informação, para obtenção
do título de Bacharel.

APROVADA em 1 de julho de 2014.

Ramon Simões Abílio

Gustavo Andrade do Vale


Heitor Augustus Xavier Costa (Orientador)

PATRÍCIA DE PAULA DIAS DE OLIVEIRA

**EXTRAÇÃO DE UMA LINHA DE PRODUTOS DE
SOFTWARE UTILIZANDO COMPILAÇÃO
CONDICIONAL**

Monografia de graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do curso de Sistemas de Informação para obtenção do título de Bacharel em Sistemas de Informação.

LAVRAS
MINAS GERAIS - BRASIL
2014

PATRÍCIA DE PAULA DIAS DE OLIVEIRA

**EXTRAÇÃO DE UMA LINHA DE PRODUTOS DE
SOFTWARE UTILIZANDO COMPILAÇÃO
CONDICIONAL**

Monografia de graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do curso de Sistemas de Informação para obtenção do título de Bacharel em Sistemas de Informação.

Área de Concentração:
Engenharia de Software

Orientador:
Prof. Dr. Heitor Augustus Xavier Costa

LAVRAS
MINAS GERAIS - BRASIL
2014

**Ficha Catalográfica preparada pela Divisão de Processo Técnico da Biblioteca
Central da UFLA**

Oliveira, Patrícia de Paula Dias de

Extração de uma Linha de Produtos de Software Utilizando Compilação Condicional / Patrícia de Paula Dias de Oliveira. Lavras - Minas Gerais, 2014. 48p.

Monografia de Graduação - Universidade Federal de Lavras. Departamento de Ciência da Computação.

1. Linha de Produtos de Software. 2. Técnicas de Extração de Linhas de Produtos de Software. 3. Compilação Condicional. I. Oliveira, P. de P. D. de. II. Universidade Federal de Lavras. III. Extração de uma Linha de Produtos de Software Utilizando Compilação Condicional.

PATRÍCIA DE PAULA DIAS DE OLIVEIRA

**EXTRAÇÃO DE UMA LINHA DE PRODUTOS DE
SOFTWARE UTILIZANDO COMPILAÇÃO
CONDICIONAL**

Monografia de graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do curso de Sistemas de Informação para obtenção do título de Bacharel em Sistemas de Informação.

Aprovada em ___/___/_____

MSc. Ramon Simões Abílio

Bel. Gustavo Andrade do Vale

Prof. Dr. Heitor Augustus Xavier Costa
(Orientador)

LAVRAS
MINAS GERAIS - BRASIL

AGRADECIMENTOS

Primeiramente agradeço a Deus por me dar força, determinação e persistência para chegar até aqui. Muitas foram às vezes em que pensei em desistir e fui fraca, mas tenho certeza de que ELE renovava minhas esperanças e fazia com que eu levantasse e continuasse a batalha. Agradeço a minha tia, Nilza, por me acolher em Lavras e dar todo suporte para que eu pudesse seguir em frente. Ao meu tio, Edson, que é meu exemplo de pessoa, profissional e pai. Obrigada por despertar e fazer viva em mim a vontade de ser uma profissional competente e dedicada, e acreditar que é possível chegar a algum lugar sendo esforçada. A minha mãe, Nilda, sem a senhora nada disso seria possível. Em todos esses 10 anos que estive longe de casa, da família, dos amigos, e principalmente da senhora, saber o quanto lutou pela minha vida quanto nasci e quanta confiança depositou em mim, foram meus pilares para que não desistisse e que quizesse cada vez mais chegar ao fim dessa etapa. Mãe tenha certeza sempre tudo que faço e procuro conquistar nunca chegará aos pés do que a senhora fez por mim, serei eternamente grata por ser sua filha e me orgulho de poder dizer que tive e tenho um exemplo de vida, mulher, mãe, pessoa, profissional e amiga. Não passará um dia em minha vida sem que eu pense e tente retribuir tudo o que fez por mim, TE AMO SEMPRE. A todos os meus familiares e amigos que de alguma forma contribuíram para a realização desse sonho. Ao meu orientador, Heitor Augustus, que depositou em mim a confiança para realização deste trabalho e me deu todo suporte para tal. Finalmente, agradeço a todos os amigos que fiz em Lavras em especial, Geraldo. Meu amigo vou carregar tudo que aprendi com você, obrigada por ter entrado na minha vida.

Extração de uma Linha de Produtos de Software Utilizando Compilação Condicional

RESUMO

Linhas de Produtos de Software (LPSs) é uma abordagem de desenvolvimento que visa a criação de uma família de sistemas de software. Apesar do crescente interesse em linhas de produtos, as pesquisas nessa área ainda são muito escassas. Isso dificulta conclusões mais amplas sobre a efetiva aplicação de princípios de desenvolvimento baseado em LPSs em sistemas reais. Dessa forma este trabalho descreve um experimento envolvendo a extração de uma linha de produtos para o TBC-GAAL, um software educacional desenvolvido na linguagem de programação Java para o ensino de Geometria Analítica e Álgebra Linear. Utilizando compilação condicional, foram implementadas dez *features* do TBC-GAAL. Acredita-se que este trabalho possa contribuir, disponibilizando referencial teórico, com outras pesquisas que visem a avaliação de técnicas, ferramentas e linguagens para implementação de linhas de produtos. Finalmente, as *features* consideradas no experimento foram caracterizadas utilizando um conjunto de medidas específicas para linhas de produtos. Diante dos resultados dessa caracterização, foram destacados os principais desafios envolvidos na extração de *features* de sistemas de software reais.

Palavras-chave: Linha de Produtos de Software, Extração de Linhas de Produtos de Software, Compilação Condicional.

Extracting a Software Product Line Using Conditional Compilation

ABSTRACT

Software Product Lines (SPLs) is a development approach that aims to create a family of software systems. Despite the increasing interest in product lines, research in this area are still very scarce. This hampers broader conclusions about the effective application of principles-based LPSs in real systems development. Thus this work describes an experiment involving the extraction of a product line for the TBC-GAAL, educational software developed in Java programming language for teaching Analytic Geometry and Linear Algebra. Using conditional compilation ten the TBC-GAAL features were implemented. It is believed that this work can contribute by providing theoretical reference with other research aimed at assessing techniques, tools and languages for implementing product lines. Finally, the features considered in the experiment were characterized using a set of specific measures for product lines. Given the results of this characterization, we highlight the key challenges involved in extracting features from real software systems.

Keywords: Software Product Line, Software Product Lines Extraction, Conditional Compilation.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

1. INTRODUÇÃO	1
1.1. Motivação	2
1.2. Objetivo	2
1.3. Metodologia de Desenvolvimento	3
1.3.1. Tipo de Pesquisa	3
1.3.2. Procedimentos Metodológicos	3
1.4. Estrutura do Trabalho	3
2. LINHA DE PRODUTOS DE SOFTWARE	5
2.1. Considerações Iniciais	5
2.2. História e Motivação.....	5
2.3. Vantagens e Desvantagens	7
2.4. Engenharia	9
2.4.1. Atividades Essenciais em LPS	9
2.4.1.1. Desenvolvimento de Ativos Base	9
2.4.1.2. Desenvolvimento do Produto.....	10
2.4.1.3. Gerenciamento	11
2.4.2. Modelos para Construção de LPS.....	12
2.5. Processo de Desenvolvimento	13
2.6. Variabilidade.....	14
2.7. Considerações Finais	15
3. EXTRAÇÃO DE LINHAS DE PRODUTOS DE SOFTWARE	17
3.1. Considerações Iniciais	17
3.2. Técnicas	17
3.2.1. Baseadas em Anotação	17
3.2.1.1. Anotações Textuais	18
3.2.1.2. Anotações Visuais	19
3.2.2. Baseadas em Composição	21
3.2.2.1. Orientação a Aspectos	21
3.2.2.2. Orientação a Características	22
3.2.2.3. Módulos de Características Aspectuais	23
3.3. Comparação das Técnicas de Extração de LPS.....	24
3.4. Considerações Finais	26
4. MEDIDAS DE SOFTWARE	28
4.1. Considerações Iniciais	28
4.2. Medidas Orientadas a Tamanho	28

4.3. Medida de Transversalidade	28
4.4. Medidas de Granularidade	29
4.5. Medidas de Localização	29
4.6. Considerações Finais	30
5. TRABALHOS RELACIONADOS	31
6. AVALIAÇÃO	33
6.1. Considerações Iniciais	33
6.2. TBC-GAAL	33
6.3. Características.....	35
6.4. Processo de Extração	35
6.5. Caracterização da LPS Extraída.....	38
6.6. Considerações Finais	41
7. CONSIDERAÇÕES FINAIS	43
7.1. Conclusões	43
7.2. Contribuições	44
7.3. Limitações.....	44
7.4. Trabalhos Futuros	44
REFERÊNCIAS BIBLIOGRÁFICAS	45

LISTA DE FIGURAS

Figura 2-1 - Atividades Essenciais no Desenvolvimento de uma LPS (Fonte: [SEI, 2004])	9
Figura 2-2 - Desenvolvimento de Ativos Base (Fonte: SEI, 2004)	10
Figura 2-3 - Desenvolvimento de Produtos (Fonte: [SEI, 2004])	11
Figura 2-4 - Modelos para Construção de LPS (Fonte: [Silva <i>et al.</i> , 2011])	12
Figura 2-5 - Processo de Desenvolvimento de LPS - Engenharia de Domínio e Engenharia da Aplicação (Fonte: [Klaus, 2005; Clements, 2002])	13
Figura 3-1 - Exemplo de Código Anotado (Fonte: [Gaia <i>et al.</i> , 2013])	19
Figura 3-2 - Exemplo de Código Colorido com CIDE (Fonte: [Couto <i>et al.</i> , 2011]) ...	20
Figura 3-3 - Exemplo de Código utilizando AOP (Fonte: [Gaia <i>et al.</i> , 2013])	22
Figura 3-4 - Exemplo de Código utilizando FOP (Fonte: [Gaia <i>et al.</i> , 2013]).....	23
Figura 6-1 - TBC-GAAL: Tela Inicial	34
Figura 6-2 - TBC-GAAL: Funções - Vetores	34
Figura 6-3 - TBC-GAAL: Funções - Cônicas e Quádricas	34
Figura 6-4 - Modelo de Características do TBC-GAAL-LPS	36
Figura 6-5 - Trechos de Código que Implementam as Características CILINDRO e OPERAÇÕES.....	37
Figura 6-6 - Exemplo de Anotação do Tipo <i>StartMethod</i>	41
Figura 6-7 - Exemplo de Anotação do Tipo <i>EndMethod</i>	41

LISTA DE TABELAS

Tabela 2-1 - Benefícios Tangíveis de uma LPS (Fonte: [Cohen, 2003]).....	7
Tabela 2-2 - Benefícios Intangíveis de uma LPS (Fonte: [Cohen, 2003])	7
Tabela 2-3 - Desvantagens de uma LPS (Fonte: [Cohen, 2003])	8
Tabela 2-4 - Problemas Identificados pelas Empresas (Fonte: [Cohen, 2003])	8
Tabela 3-1 - Tabela Comparativa	25
Tabela 6-1 - Medidas de Tamanho para os Produtos.....	38
Tabela 6-2 - Medidas de Tamanho para as Características	39
Tabela 6-3 - Medida de Transversalidade: <i>Grau de Espalhamento (SD)</i>	40
Tabela 6-4 - Medida de Granularidade: <i>Class</i>	40
Tabela 6-5 - Medida de Localização: <i>StartMethod</i> e <i>EndMethod</i>	40

1. INTRODUÇÃO

A maneira com que as indústrias desenvolvem seus produtos e seus serviços mudou de forma significativa no decorrer dos anos. Por causa da concorrência acirrada e da diversificação dos produtos ofertados, os consumidores tornaram-se mais exigentes [Bayer *et al.*, 1999]. Novas tecnologias fazem com que os produtos e os serviços ofertados atendam a diversas necessidades dos consumidores. Para atender as expectativas dos clientes e serem competitivas, as empresas precisam buscar metodologias e ferramentas que a auxiliem. Além disso, *stakeholders* procuram produtos com funções específicas para atender suas demandas.

Um exemplo que aborda de maneira simples o cenário exposto anteriormente é a indústria automobilística. No início da revolução industrial, a produção era em larga escala (em massa) e os carros eram padronizados; a princípio, não havia diferenciação entre os modelos e as cores dos veículos [Bayer *et al.*, 2000]. Os clientes começaram a procurar carros que atendessem as suas necessidades, pois uma família precisava de um carro com lugares suficientes para transportar todos; por outro lado, uma pessoa solteira não precisava de um carro com vários lugares. Dessa forma, a indústria foi adequando-se a essas novas tendências até chegar a configuração que existe hoje: carros personalizados de acordo com o perfil de cada cliente [Flege *et al.*, 1998]. É importante ressaltar que, para garantir a qualidade dos carros e alinhar a relação custo/benefício, é essencial ter um processo produtivo adequado e que acompanhe o desenvolvimento tecnológico.

A indústria de desenvolvimento de software não é diferente da automobilística. É importante que sistemas de software iguais possam ser utilizados em diferentes empresas e serem moldados de acordo com as necessidades de cada uma. Nesse sentido, a indústria de desenvolvimento de software buscou alternativas para adequar-se às novas tendências, alinhando aspectos relativos ao desenvolvimento de produtos com qualidade, de maneira ágil, em grande escala e de forma personalizada. A abordagem Linha de Produto de Software - LPS (do inglês *Software Product Lines*, SPL) surge como alternativa para alcançar esses aspectos [Almeida *et al.*, 2005].

Porém, com essa abordagem, diversas barreiras surgem em relação à sua implantação, por exemplo, custo, retorno para a empresa e como e quando será realizada a

implantação [Knodel *et al.*, 2006]. Cabe ressaltar que devem ser utilizadas ferramentas e técnicas para auxiliar na extração de uma LPS. O principal objetivo da abordagem LPS é migrar para uma cultura de desenvolvimento em que novos sistemas são derivados a partir de um conjunto de componentes e de artefatos comuns, os quais constituem o núcleo da LPS [Knodel *et al.*, 2006]. Técnicas de extração consistem em implementar a variabilidade da LPS.

1.1. Motivação

A motivação em realizar este trabalho é em decorrência da complexidade na implantação de uma LPS [Almeida *et al.*, 2005; Knodel *et al.*, 2006]. O alto investimento inicial, a resistência organizacional e o retorno, muitas vezes, em longo prazo, dificultam a implementação dessa abordagem; por outro lado, depois de implementada, o retorno que uma LPS pode trazer compensa os investimentos iniciais [Silva; Neto; Garcia; Muniz, 2011]. Assim, o processo de implantação de LPSs torna-se uma tarefa difícil [Almeida *et al.*, 2005; Knodel *et al.*, 2006]. Dessa forma, a escolha de uma técnica de extração com suporte ferramental torna-se necessária. Para implementar uma LPS, pode ser utilizada a Compilação Condicional e medidas de software podem ser empregadas para analisar os benefícios que essa abordagem pode trazer para a empresa.

1.2. Objetivo

Neste trabalho, o objetivo é extrair uma LPS de um software existente e realizar medições para identificar benefícios. Para extrair a LPS, foi utilizada a Compilação Condicional, baseada em anotação de código. Na fase de análise da LPS extraída, foram utilizadas medidas de software para obter dados quantitativos da LPS extraída. Para atingir esse objetivo, foi necessário alcançar os seguintes objetivos específicos:

- Realizar estudo sobre Compilação Condicional, buscando aprimorar os conhecimentos;
- Obter informações sobre o software a ser utilizado na extração da LPS;
- Realizar busca na literatura sobre medidas de software a serem abordadas para realizar a análise e obter as características e os benefícios da LPS extraída.

1.3. Metodologia de Desenvolvimento

1.3.1. Tipo de Pesquisa

Quanto à natureza, este trabalho pode ser classificado como **pesquisa aplicada**, pois a LPS extraída pode servir como exemplo para outras pesquisas. Quanto aos objetivos, pode ser caracterizado como **pesquisa exploratória**, pelo fato de ser extraída uma LPS de um software real. Quanto à sua abordagem, este trabalho pode ser uma **pesquisa qualitativa**, pois expõe um conjunto de referencial teórico, e **pesquisa quantitativa**, pois utiliza medidas para avaliar a LPS extraída. Em relação aos procedimentos, pode ser caracterizado como **estudo de caso**, pois foram obtidos dados relevantes em relação à adoção da abordagem de LPS e realiza o estudo um caso específico, software TBC-GAAL. A coleta de dados é feita por meio de **observação**, pois é um estudo distinto em relação à extração de uma LPS do software específico.

1.3.2. Procedimentos Metodológicos

Este trabalho foi realizado no período de janeiro de 2013 a julho de 2014, iniciando-se por um levantamento bibliográfico, na internet, em bibliotecas digitais e impressas e artigos científicos relacionados ao tema. Os primeiros esforços foram estudar o conceito, a história e a importância da abordagem LPS. Foi realizada uma busca da origem do termo e a aplicabilidade de LPS nas indústrias de desenvolvimento de software e academia. Posteriormente, técnicas de extração foram estudadas para identificar seus critérios de caracterização e realizar uma análise comparativa entre elas. Com essa análise, foi possível obter embasamento teórico sobre a técnica utilizada neste trabalho, Compilação Condicional. Em seguida, foi escolhidas medidas de software utilizadas para caracterizar LPS. Essas medidas foram brevemente apresentadas e contextualizadas.

Um software foi escolhido para realizar a extração da LPS e algumas medidas foram utilizadas para avaliar o resultado da extração. Os resultados obtidos foram analisados para comparar a LPS extraída em relação ao software original.

1.4. Estrutura do Trabalho

O restante deste trabalho está organizado da seguinte forma.

Breve revisão da literatura sobre LPSs, expondo conceitos, história e motivação, vantagens e desvantagens em adotar essa abordagem, processo de desenvolvimento e contextualização sobre variabilidades, é apresentada no Capítulo 2.

Técnicas utilizadas para realizar a extração de uma LPS, abordando principais características, são discutidas no Capítulo 3.

Algumas medidas utilizadas para avaliar a LPS extraída são tratadas no Capítulo 4.

Trabalhos semelhantes estão resumidos no Capítulo 5.

A extração de uma LPS, a partir de um software existente, e sua avaliação utilizando algumas medidas são apresentadas no Capítulo 6.

Conclusões, contribuições, limitações e sugestões de trabalhos futuros são discutidas no Capítulo 7.

2. LINHA DE PRODUTOS DE SOFTWARE

2.1. Considerações Iniciais

O mercado de desenvolvimento de software precisa construir produtos com melhor qualidade, redução de custos, adaptação rápida a mudanças e menor tempo de colocação no mercado, atendendo às necessidades dos clientes. Visando a essas expectativas, vários esforços em criar processos e arquiteturas de sistemas têm sido realizados ao longo dos anos. Seguindo esse caminho, uma nova abordagem para reutilização de software tem ganhado atenção da indústria e da academia, conhecida como Linha de Produtos de Software (LPS).

Breve histórico e motivação, para mostrar como surgiu a ideia de LPS e porque é uma abordagem que ganha força e espaço no mercado de desenvolvimento de software, é apresentado na Seção 3.2. Vantagens e desvantagens em adotar LPS são discutidas na Seção 3.3. Atividades envolvidas na LPS são abordadas na Seção 3.4. Os processos Engenharia de Domínio e Engenharia de Aplicação são destacados na Seção 3.4. Conceitos relacionados a variabilidades em LPS são mostrados na Seção 3.5.

2.2. História e Motivação

Com o surgimento do desenvolvimento orientado a objetos na década de 80, a comunidade de software identificou a oportunidade de reutilização de código [Durscki, 2004]. No entanto, a necessidade de desenvolver software com mais eficiência foi notada apenas na década de 90, quando várias experiências com reuso de software confirmaram que, sem o planejamento adequado, desenvolver artefatos a partir do zero poderia ser menos custoso do que reutilizar software. Dessa forma, torna-se fundamental para o sucesso do projeto realizar o planejamento dos produtos que poderão ser gerados a partir do domínio e das características pertencentes a esses produtos [Klaus, 2005].

No mercado de desenvolvimento de software, é possível identificar características semelhantes aos tipos de produtos individuais e os produzidos em massa, denominados software individual e software padrão, respectivamente. No entanto, software individual é caracterizado pelo custo elevado, pois o esforço para desenvolver características específicas é relativamente maior do que o desenvolvimento de um produto de softwre

padrão. Em contrapartida, em software padrão, ocorre a falta de diversificação suficiente para atender as expectativas dos clientes [Silva *et al.*, 2011].

Diante da elevada e crescente necessidade de desenvolver software com mais individualidade, atendendo a requisitos específicos de cada *stakeholder*, as indústrias precisam encontrar alternativas para reduzir os custos de desenvolvimento software individual e manter-se competitiva [Schmid, 2007]. LPS mostrou-se como uma maneira interessante em lidar com as necessidades do usuário e com as restrições de recursos variados, embora o foco seja a derivação eficiente de variantes de produtos personalizados [Figueiredo *et al.*, 2008]. Na literatura, são encontradas várias definições para LPS, sendo uma delas [Clements *et al.*, 2002]

Um conjunto de sistemas de software que compartilham um conjunto comum e gerenciado de funções (*feature*¹) que satisfazem as necessidades específicas de um segmento, desenvolvidos a partir de um aglomerado comum de ativos base de forma planejada.

Os ativos base da LPS representam o conjunto mínimo de funções que refletem o domínio. Porém, deseja-se desenvolver sistemas de software cada vez mais customizáveis; dessa forma, surge o conceito de *features* (características), conjunto de artefatos de software que adicionam funções ao código fonte de uma LPS [Ferreira, 2012; Kästner, 2007]. Geralmente, as características de um sistema de software são documentadas em um Modelo de Características, que permite visualizar as possibilidades de produtos instanciáveis [Kang *et al.* 1990].

O reúso de software possui conceitos diferentes de LPS; no reúso, as organizações possuem repositórios contendo bibliotecas de componentes, de módulos e de algoritmos utilizados pelos desenvolvedores. Geralmente, a dificuldade em reusar é o tempo gasto para identificar as funções desejadas e adaptá-las à aplicação atual ser maior do que para construí-las novamente [Jan, 2000].

As LPSs podem ajudar as organizações a superar os problemas causados pela escassez de recursos. Organizações descobriram que uma LPS, quando habilmente implementada, pode produzir benefícios, incluindo vantagens competitivas [Durscki *et al.*,

¹ *Features* são componentes que adicionam funcionalidades ao código base de uma linha de produtos [Kästner, 2007].

2004]. Porém, as dificuldades para implantar uma LPS são diversas e podem vir de várias fontes, por exemplo, resistência organizacional, gerencial e dos desenvolvedores; por outro lado, há benefícios, por exemplo, ganho em produtividade [Cohen, 2003]. A implantação e a institucionalização de uma LPS são custosas e, em consequência, demanda cuidado e planejamento [Durscki *et al.*, 2004]. Portanto, para adotar LPS, é importante o administrador conhecer a metodologia e realizar estudo criterioso para avaliar a viabilidade de sua aplicação na empresa.

2.3. Vantagens e Desvantagens

Diante de um mercado cada vez mais acirrado, em que a busca por qualidade, melhor desempenho e redução nos custos são requisitos essenciais para o desenvolvimento de produtos, as LPSs surgem como alternativa para obter resultados de maneira eficaz. Para mapear as vantagens, foram classificadas em duas categorias [Cohen, 2003]:

- **Tangíveis.** Benefícios medidos diretamente, como redução de defeitos e aumento da lucratividade (Tabela 2-1);
- **Intangíveis.** Benefícios relatados pelos envolvidos, não são avaliados por meio de medidas, como satisfação do cliente e dos desenvolvedores (Tabela 2-2).

Tabela 2-1 - Benefícios Tangíveis de uma LPS (Fonte: [Cohen, 2003])

Benefícios Tangíveis	
Lucratividade	Os ativos base permitem que a organização produza produtos aos vários segmentos de mercado. Isso ocasiona aumento de sua fatia no mercado e maior rentabilidade.
Qualidade	Em sistemas desenvolvidos em LPSs, é relatada redução na quantidade de defeitos. A qualidade pode ser medida em termos de redução no tempo de correções por serem tratadas localmente não havendo efeito cascata.
Desempenho dos Produtos de Software	O desempenho dos produtos de software é mensurado em relação ao nível de maturidade da LPS. Como os ativos base da LPS estão em constante otimização esse nível de maturidade aumenta.
Tempo de Integração	Por o desenvolvimento ser incremental, o tempo de integração é reduzido.
Produtividade	A equipe de desenvolvimento pode ser reduzida, o custo de desenvolvimento diminui, o cronograma é reduzido e o sistema possui flexibilidade documentada o que facilita o atendimento das solicitações de modificações do cliente.

Tabela 2-2 - Benefícios Intangíveis de uma LPS (Fonte: [Cohen, 2003])

Benefícios Intangíveis	
Desgaste de Profissionais	Diminuição da rotatividade de profissionais em relação ao desenvolvimento tradicional.
Aceitação dos Desenvolvedores	Após treinamento inicial, os desenvolvedores relatam satisfação em trabalhar com a abordagem baseada em ativos e arquitetura comuns.
Satisfação Profissional	Desenvolvedores relatam que o trabalho braçal foi realizado (desenvolvimento dos ativos base). Eles podem concentrar-se em atividades mais interessantes, como o aperfeiçoamento/inação de elementos específicos.
Satisfação do Cliente	Ativos reduzem os riscos, aumentando a previsibilidade da entrega e diminuindo a taxa de defeitos. Esses fatores afetam positivamente o cliente (induzindo-o a preferir produtos derivados de LPSs).

Com esses benefícios, pode ser difícil imaginar desvantagens na implantação de LPS. Porém, sua institucionalização e sua implementação são custosas e demandam alto investimento inicial, que talvez seja uma das principais limitações para a adoção dessa abordagem. O processo de implantação de uma LPSs envolve pessoas, que devem ser sensibilizadas e treinadas para obter sucesso com essa abordagem (pode ser difícil de alcançar), e tecnologias, que podem ter custo elevado, sendo necessário monitorá-las e controlá-las. Problemas foram identificados na implantação de LPSs (Tabela 2-3), por exemplo, falta de um líder comprometido; em organizações que implataram LPS, foram "reforçados" a presença de alguns desses problemas (Tabela 2-4) [Cohen, 2003].

Tabela 2-3 - Desvantagens de uma LPS (Fonte: [Cohen, 2003])

Desvantagens	
Falta de um Líder Comprometido	A presença de um líder em projeto de implantação da LPS é importante, pois ele é responsável por manter a equipe motivada com foco na abordagem e auxiliar os desenvolvedores. Portanto, deve ser uma pessoa que acredita nos princípios do modelo e supervisiona o processo. Caso ela não receba a autoridade necessária, não acredita no modelo ou não exista, é difícil manter o foco e a confiança no processo havendo risco de desistência.
Falta de Compromisso da Gerência	Como essa abordagem envolve grande investimento inicial e o retorno é relativamente lento, é importante a gerência estar convencida da viabilidade e das vantagens que o projeto trará à organização.
Abordagem Inadequada	A LPS deve estar alinhada aos objetivos estratégicos da organização. Apesar de seus objetivos variarem de empresa para empresa, eles estão baseados no ganho de produtividade, de custo e de qualidade na exploração das similaridades de produtos. Caso os produtos planejados para a LPS não possuam similaridades para garantir a viabilidade da abordagem, a LPS pode não trazer o retorno planejado.
Falta de Compromisso da Equipe	O líder deve supervisionar e orientar a equipe. Um dos principais pontos é fazer com que a equipe desenvolvedora tenha compromisso com os princípios e com o processo de desenvolvimento da LPS. Uma equipe desmotivada e que não acredita na abordagem impacta negativamente o processo. A maneira de evitar esse problema é envolver membros da equipe nas atividades de elaboração da LPS.
Interação Insuficiente entre as Equipes	Em um projeto de desenvolvimento de uma LPS, é inevitável a interação entre os setores (equipe) como gerência, <i>marketing</i> e engenharia. Falta de interação e de colaboração entre essas equipes pode impedir a obtenção dos objetivos planejados.
Padronização Desapropriada	As organizações, em muitos casos, assumem padrões para guiar automaticamente a implantação de uma LPS. Porém, esses padrões devem ser estudados caso a caso e, apesar de serem importantes para a LPS, devem ser discutidos antes de sua implementação. Padrões definidos aleatoriamente sem estudo criterioso pode trazer prejuízos à organização como limitação nas opções de tecnologia da LPS.
Adaptação Insuficiente	As práticas organizacionais devem ser alinhadas com as características da LPS para que elas essas práticas permitam adaptação ou personalização e não ocorram imprevistos nas atividades envolvidas no processo da LPS.
Evolução da Abordagem	Para acompanhar as novas necessidades e tendências, deve-se estar em constante melhoria. Caso o processo da LPS não seja revisado e atualizado periodicamente, suas práticas tornam-se obsoletas e inapropriadas e comprometem a LPS.
Falha de Disseminação	O líder é responsável por desenvolver e distribuir a documentação em nível e em tipo apropriados para a organização, treinar os envolvidos e dar o apoio necessário. Caso falhe, pode prejudicar o cronograma e os objetivos da LPS. Essas pessoas devem ser escolhidas de forma criteriosa e cuidadosamente.

Tabela 2-4 - Problemas Identificados pelas Empresas (Fonte: [Cohen, 2003])

Problema	Percentual Identificado
Resistência organizacional	52%
Resistência gerencial	36%
Resistência dos desenvolvedores	32%

Tabela 3 4 - Problemas Identificados pelas Empresas (Fonte: [Cohen, 2003]) (cont.)

Problema	Percentual Identificado
Preocupações e desconfianças com o tamanho do investimento	45%
Falta de recursos devidamente treinados e capacitados	29%
Incapacidade de analisar e prever o impacto da implantação	19%
Preocupação sobre o longo tempo para retorno do investimento	18%

2.4. Engenharia

2.4.1. Atividades Essenciais em LPS

No desenvolvimento de uma LPS, há três atividades essenciais e interativas que misturam práticas de negócio e de tecnologia (Figura 2-1). Na primeira atividade, **Desenvolvimento de Ativos Base**, o foco não é criar o produto final, pois seu objetivo é desenvolver ativos a serem utilizados por atividades posteriores. Na segunda atividade, **Desenvolvimento do Produto**, há reuso de ativos desenvolvidos na fase anterior. Na terceira atividade, **Gerenciamento**, a gestão técnica e organizacional é abordada. As setas rotatórias indicam que as atividades estão conectadas umas com as outras e alterações podem ser realizadas de acordo com o desenvolvimento da LPS. A atividade Gerenciamento é importante e está presente em todo ciclo, pois, em qualquer projeto de desenvolvimento, o acompanhamento das atividades deve ser constante para que elas sejam desempenhadas da melhor maneira.



Figura 2-1 - Atividades Essenciais no Desenvolvimento de uma LPS (Fonte: [SEI, 2004])

2.4.1.1. Desenvolvimento de Ativos Base

Os ativos base correspondem a um conjunto de elementos customizáveis, utilizados na construção de produtos de software, em conjunto compõem a plataforma da LPS [Durscki *et al.*, 2004; Silva *et al.*, 2011]. Nessa atividade, o objetivo é definir os elementos

comuns e a variabilidade da LPS, obtendo artefatos reutilizáveis e aumentando a capacidade de produção. Os componentes de software, artefatos, *design patterns*, documentação dos requisitos comuns à família de produtos, a arquitetura da LPS e cronogramas são exemplos de ativos base que podem ser utilizados em produtos sem adaptações [Durscki *et al.*, 2004]. Porém, algumas adaptações permitem que eles tornem-se mais utilizáveis na LPS, as quais devem ser planejadas antes do desenvolvimento e facilitadas para a equipe de desenvolvimento do produto sem colocar em risco as propriedades existentes dos ativos base.

Na Figura 2-2, as setas rotatórias mostram a existência de um movimento para adicionar restrições ou novos padrões no desenvolvimento que afetam diretamente as saídas do processo. Em alguns casos, os ativos base podem ser desenvolvidos do zero ou a partir de produtos existentes. Essa atividade possui como principais artefatos de saída o **Domínio da LPS** (descrição dos produtos que constituirão a LPS ou é capaz de produzir), os **Ativos Base da LPS** (base para a produção dos produtos da LPS) e o **Plano de Produção dos Produtos** (descrição das decisões a serem tomadas para instanciar produtos específicos a partir dos ativos base da LPS) [Edson, 2005].



Figura 2-2 - Desenvolvimento de Ativos Base (Fonte: SEI, 2004)

2.4.1.2. Desenvolvimento do Produto

O processo de desenvolvimento de produtos de software utilizando uma LPS é realizado com planos de produção a partir dos ativos base. Ao definir um plano de produção, devem ser identificados quais ativos base fazem parte do produto e estabelecido um vínculo aos processos anexos de cada ativo base utilizado [Durscki *et al.*, 2004; Ahmed

et al., 2009]. De posse do plano de produção, o engenheiro de software pode montar as partes da LPS.

Na Figura 2-2, as entradas essenciais para o processo de desenvolvimento do produto são os **Requisitos**, o **Domínio da LPS** e os **Ativos Base**. Esse conjunto de entradas compõe o **Plano de Produção**. As setas rotatórias indicam iterações entre as partes envolvidas. A gestão deve estar presente no processo de desenvolvimento, pois garante a coordenação da infraestrutura necessária.

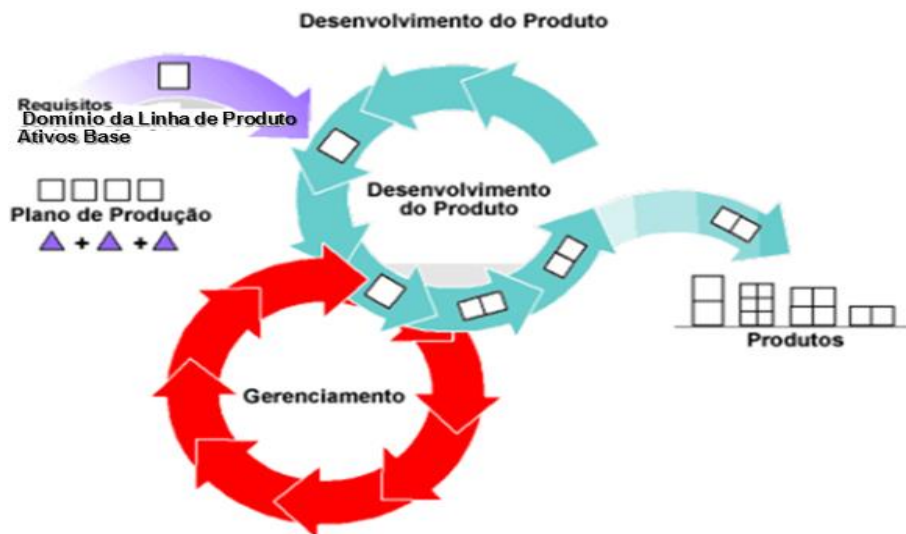


Figura 2-3 - Desenvolvimento de Produtos (Fonte: [SEI, 2004])

2.4.1.3. Gerenciamento

Nessa atividade, é fornecida e coordenada a infraestrutura necessária e possui atividades para apoiar o ciclo de vida do processo, o que representa papel essencial no sucesso da implantação de uma LPS dentro da organização [Ahmed *et al.*, 2009]. Essa atividade pode ser dividida em duas categorias [SEI, 2004]:

- **Gerenciamento Técnico.** Coordenação dos ativos base e desenvolvimento do produto, garantindo que as equipes de desenvolvimento sigam os processos definidos para a LPS;
- **Gerenciamento Organizacional.** Garantia das unidades organizacionais receberem os recursos corretos em quantidades suficientes.

O conjunto de ativos base e os planos de produção não são construídos sem estudar o ambiente, a viabilidade do projeto, a caracterização do negócio entre outros fatores; nesse contexto, o investimento organizacional é indispensável. A utilização dos ativos base deve

ser guiada, controlada e garantida pelo gerenciamento. As LPSs estão mais relacionadas às práticas de negócios do que práticas técnicas [Silva *et al.*, 2011].

2.4.2. Modelos para Construção de LPS

Há três modelos predominantes para a construção de LPSs [Chen *et al.*, 2005] (Figura 2-4):

- **Proativo.** Possui semelhanças ao modelo de ciclo de vida em cascata. Nesse modelo, cada atividade deve ser finalizada antes que a próxima atividade possa ser iniciada. Dessa maneira, uma LPS é projetada e implementada para apoiar o escopo dos sistemas necessários. A LPS projetada tem como base os possíveis sistemas a serem desenvolvidos futuramente. Esse modelo requer investimento antecipado nos ativos de produção e é apropriado quando os requisitos para o conjunto de produtos a serem criados são estáveis e podem ser definidos antecipadamente;
- **Reativo.** Propõe que a LPS seja desenvolvida incrementalmente conforme a demanda para novos produtos ou quando surgem novos requisitos para os produtos existentes. Esse modelo é adequado para produtos em que não é possível prever os requisitos de suas variações;
- **Extrativo.** Baseia-se na ideia que uma LPS é gerada a partir da extração de características (*features*) do código fonte original de um sistema base. É uma abordagem apropriada quando existe a disponibilidade de um ou mais sistemas com finalidade semelhante para reutilizar e entre eles existem diferenças que caracterizam funções relevantes aos usuários.

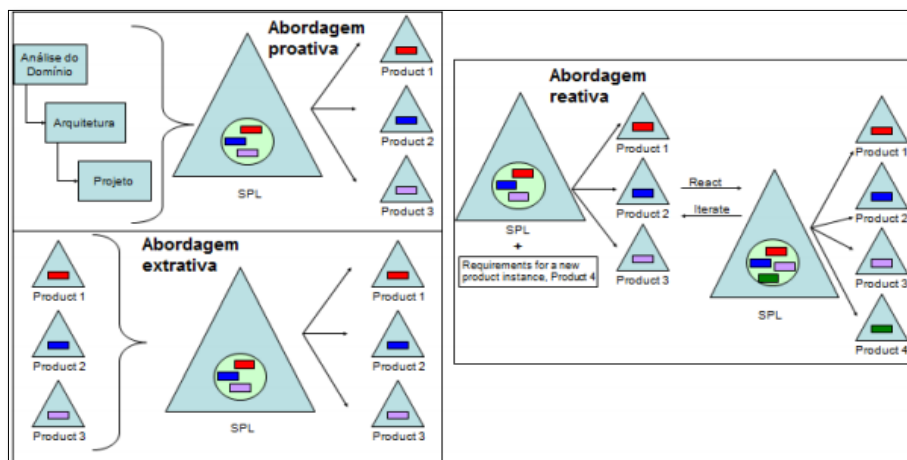


Figura 2-4 - Modelos para Construção de LPS (Fonte: [Silva *et al.*, 2011])

2.5. Processo de Desenvolvimento

O processo de desenvolvimento de uma LPS é dividido em duas partes [Klaus, 2005; Clements, 2002]:

- **Engenharia de Domínio.** Responsável por estabelecer a plataforma de reutilização e definir a comunalidade e a variabilidade da LPS. A plataforma consiste nos tipos de artefatos de software (requisitos, design, testes, etc.) chamados de ativos base;
- **Engenharia de Aplicação.** Responsável por derivar aplicações concretas a partir da plataforma estabelecida na Engenharia de Domínio. Ela explora a variabilidade da LPS e assegura sua correta instanciação de acordo com as necessidades específicas das aplicações finais.

A vantagem dessa divisão é a separação de objetivos para construir uma plataforma robusta e para construir aplicações específicas em um curto espaço de tempo. Para serem eficazes, as duas partes devem interagir de maneira que seja benéfica para ambos. Os dois processos são apresentados na Figura 2-5.

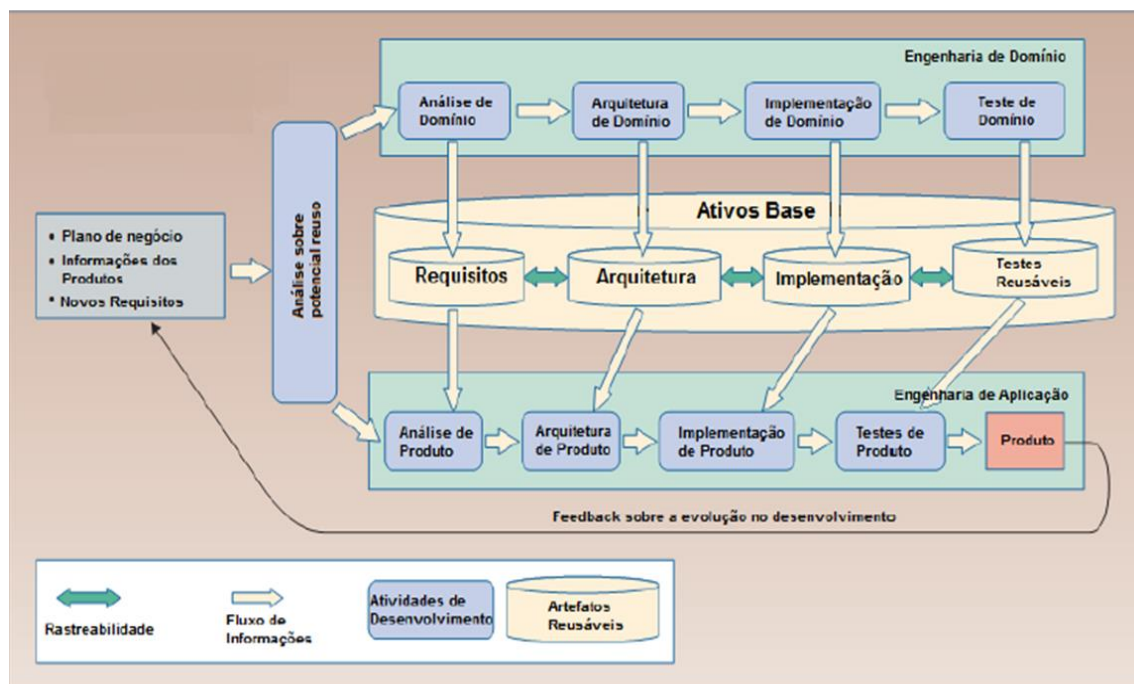


Figura 2-5 - Processo de Desenvolvimento de LPS - Engenharia de Domínio e Engenharia da Aplicação (Fonte: [Klaus, 2005; Clements, 2002])

Analisando a Figura 2-5, os ativos base são os artefatos e os recursos que formam a base da LPS, incluem requisitos, arquitetura, implementação e testes reusáveis. A Engenharia de Domínio é composta pelas fases **Análise do Domínio** (entendimento e modelagem do domínio), **Arquitetura do Domínio** (especificação da arquitetura),

Implementação do Domínio (requisitos, projeto e construção dos componentes do domínio) e **Teste de Domínio**. A Engenharia de Aplicação é o processo de construção de sistemas a partir dos artefatos obtidos na Engenharia de Domínio. As aplicações são criadas a partir dos artefatos comuns e de um subconjunto dos artefatos variáveis desenvolvidos na Engenharia de Domínio, sendo composta pelas fases **Análise de Produto, Arquitetura de Produto, Implementação de Produto e Teste de Produto**.

2.6. Variabilidade

Entre os vários aspectos a serem gerenciados em uma LPS, há as diferenças entre seus produtos, conhecidas como variabilidades. Variabilidade é um dos assuntos relacionados à LPS que mais chama a atenção por causa da complexidade de seu gerenciamento. De maneira simples, o conceito de variabilidade está relacionado às possibilidades de mudança ou de personalização de um sistema de software [Silva *et al.*, 2011; Heymans; Trigaux, 2003].

Na literatura, alguns trabalhos relatam como criar e instanciar ativos base da LPS, tais como, modelos de domínio e arquiteturas para gerar instâncias específicas; no entanto, tem sido dada pouca atenção em como realmente lidar com variabilidade no código fonte [Bayer *et al.*, 2000; Anastasopoulos *et al.*, 2000; Gacek; Vukovic, 2000]. Variabilidades e suas regras de composição devem ser refletidas no código e serem instanciadas juntamente com a criação de instâncias de cada produto [Anastasopoulos; Gacek, 2001]. A variabilidade é descrita por pontos de variabilidade e variantes. Um ponto de variabilidade é um local específico de um artefato em que uma decisão de projeto não foi resolvida. Cada ponto de variabilidade está associado um conjunto de variantes, o conjunto de variabilidades utilizado para instanciá-los e o tempo de ativação dos variantes. Cada variante corresponde a uma alternativa de projeto para instanciar determinada variabilidade [Heymans; Trigaux, 2003; Silva *et al.*, 2011; Gacek; Vukovic, 2000].

Um sistema de software passa por diversas fases de desenvolvimento que contém suas próprias representações que podem ser consideradas como diferentes níveis de abstração do sistema. Os pontos de variabilidade podem ser apresentados em vários níveis de abstração, por exemplo, descrição da arquitetura, documentação por diagramas, código fonte, código compilado, código ligado e código de execução [Silva *et al.*, 2011]. Os pontos de variabilidade mais importantes precisam ser identificados antecipadamente. No

entanto, não é trivial adaptar uma arquitetura existente para suportar certo ponto de variabilidade [Anastasopoulos; Gacek, 2001]. Dessa forma, surge o gerenciamento de variabilidades, uma tarefa importante que consiste em diretrizes para reduzir os impactos relacionados à identificação e à inserção de novos pontos de variabilidade e variantes. Algumas tarefas foram propostas para o gerenciamento de variabilidades [Anastasopoulos; Gacek, 2001; Babar *et al.*, 2010]:

- **Identificar variabilidades.** Com os requisitos, os desenvolvedores formam uma especificação adequada para a LPS. O objetivo é identificar as diferenças entre os produtos e quais características são compartilhadas por eles (domínio);
- **Introduzir variabilidade no sistema.** Essa tarefa depende do nível em que a variabilidade é introduzida, o tempo em que o sistema é ligado a uma variante particular e a forma que novos variantes (caso haja) são adicionados ao sistema;
- **Agrupar variantes.** Nessa tarefa, um conjunto de variantes é associado a um ponto de variabilidade. A coleção de variantes pode ser implícita ou explícita e aberta ou fechada. Caso seja implícita, desenvolvedores/usuários escolhem variantes adequados. Uma coleção explícita implica que o sistema pode decidir qual variante utilizar. Se uma variante não puder ser adicionada, então a coleção é fechada; caso adições possam ser feitas, então a coleção é aberta;
- **Vincular o sistema a uma variante.** De acordo com a perspectiva do sistema, essa vinculação pode ser feita interna ou externamente. Em uma ligação interna, o sistema possui a capacidade de vincular uma variante particular; em contrapartida, se a ligação é realizada externamente, o sistema tem que utilizar outras ferramentas, como as de gerenciamento de configuração, para executar a vinculação. Essa vinculação resulta em um sistema em que um ponto de variabilidade particular é associado a uma variante.

2.7. Considerações Finais

Neste capítulo, foram abordados conceitos de LPS, propiciando uma visão geral sobre o assunto. Foi visto que as organizações podem obter benefícios de negócio em termos de, por exemplo, redução de custos, redução do tempo de colocação no mercado e aumento na qualidade do produto, utilizando a abordagem de LPSs. Porém, há riscos e custos associados à adoção dessa abordagem que devem ser avaliados.

Foram apresentadas as atividades essenciais para a implantação de LPSs, destacando a atividade de gestão, presente no processo. Além disso, foram abordados os processos de desenvolvimento, exemplificando a Engenharia de Domínio e a Engenharia de Aplicação. Finalmente, foi apresentado um conceito importante em LPSs: variabilidade, que define as particularidades de uma LPS, sendo sua gestão importante.

LPS é uma abordagem relativamente nova que vem ganhando bastante atenção da indústria e da academia. No entanto, é necessário realizar um estudo criterioso para identificar as técnicas e as metodologias mais adequadas para o contexto da organização. O apoio da alta gerência é essencial para adotar essa abordagem. Dessa forma, diferentes contextos e aplicações devem ser analisados antes de implantar uma LPS em uma organização de desenvolvimento de software.

3. EXTRAÇÃO DE LINHAS DE PRODUTOS DE SOFTWARE

3.1. Considerações Iniciais

Na implantação de LPSs, além da preocupação com a escolha do modelo de construção de LPSs, é importante identificar o mecanismo para gerenciamento e extração de características que mais se adapta ao escopo da organização. Neste trabalho, esses mecanismos são considerados técnicas de extração, que consistem em identificar os produtos da LPS com auxílio de ferramentas e procedimentos.

Breve descrição de algumas técnicas de extração de LPS, abordando os procedimentos adotados, é apresentada na Seção 4.2. Uma análise comparativa das técnicas é discutida na Seção 4.3.

3.2. Técnicas

Na literatura, podem ser identificadas algumas técnicas utilizadas na extração de LPSs organizadas em dois grupos: i) baseadas em anotação; e ii) baseadas em composição. As técnicas baseadas em anotação podem ser divididas em anotações textuais e anotações visuais [Ahmed *et al.*, 2009].

3.2.1. Baseadas em Anotação

As tecnologias baseadas em anotação propõem que o código fonte das características da LPS mantenha-se entrelaçado ao código base do sistema, sendo a identificação dessas características feita por meio de anotações explícitas. Anotações explícitas são aquelas em que há a necessidade de acrescentar metainformações diretamente no código fonte do software para delimitar o código a ser analisado por um pré-processador. Um pré-processador é um programa automaticamente executado antes do compilador com a finalidade de examinar o código fonte a ser compilado e, de acordo com as metainformações nele adicionadas, deve executar modificações antes de repassá-lo ao compilador para tornar o programa executável [Gaia, 2013; Couto *et al.*, 2011].

As anotações explícitas podem ser do tipo textual e do tipo visual. Anotações textuais são aquelas em que o código fonte é anotado por meio da utilização de diretivas especiais entendidas pelo pré-processador, tais como as diretivas de compilação `#ifdef` e `#endif`, utilizadas pelas linguagens C/C++ [Silva *et al.*, 2011]. Anotações visuais são

aquelas em que o trecho de código a ser pré-processado é destacado com auxílio de cores diferentes para cada característica. Para tanto, é utilizada uma camada de visualização de uma IDE (*Integrated Development Environment*) [Couto *et al.*, 2011].

3.2.1.1. Anotações Textuais

A técnica conhecida por pré-processamento baseia-se na utilização de diretivas de compilação para informar a um pré-processador quais trechos de código devem ser incluídos/excluídos da compilação do software. Diretivas de compilação correspondem a linhas de código não compiladas, sendo dirigidas ao pré-processador, chamado pelo compilador antes do início do processo de compilação propriamente dito. Portanto, o pré-processador modifica o código fonte, entregando ao compilador um programa modificado de acordo com as diretivas analisadas [Couto *et al.*, 2011].

A Compilação Condicional é um mecanismo de implementação e de gerenciamento de variabilidades em LPSs que utiliza anotação de código [Gaia, 2013]. Basicamente, diretivas de pré-processamento são utilizadas para delimitar linhas do código fonte que devem ou não ser incluídas em uma versão do software [Gaia, 2013; Santos; Valente, 2008]. Ela foi utilizada vários anos em linguagens como C e é suportada em linguagens orientadas a objeto como C++. Em Java, não há suporte nativo a Compilação Condicional, porém há ferramentas que fornecem suporte, como Antenna² e Javapp³. Essa técnica não extrai fisicamente o código das características, pois as anotações são feitas no próprio código, dificultando a visualização e a identificação das características além de diminuir a legibilidade do código [Couto *et al.*, 2011].

Como são feitas anotações no código, algum mecanismo deve ser utilizado para identificar as diretivas de compilação. Assim, a Compilação Condicional utiliza um símbolo especial (`// #`) [Santos; Valente, 2008]. Os exemplos mais conhecidos de diretivas de compilação são `#ifdef`, `#ifndef`, `#else`, `#elif` e `#endif` utilizadas pelos pré-processadores das linguagens C e C++ [Couto *et al.*, 2011]. As diretivas `#ifdef` e `#ifndef` são utilizadas para marcar o início de um bloco de código a ser compilado, caso as condições associadas a essas diretivas sejam atendidas. As diretivas `#else` e `#elif` demarcam o bloco de código a ser compilado, caso o resultado das expressões associadas à

² <http://antenna.sourceforge.net/>

³ <http://www.slashdev.ca/javapp/>

diretiva `#ifndef` seja falso. A diretiva `#endif` é utilizada para delimitar o fim do bloco de código anotado [Gaia, 2013; Santos; Valente, 2008; Couto *et al.*, 2011].

Na Figura 3-1, é apresentado um trecho de código, cujo objetivo é cadastrar as ações de redirecionamento de páginas, utilizando diretivas de pré-processamento. Por exemplo, na linha 4, existe uma diretiva utilizada no pré-processamento de um trecho de código pertencente a característica `Bankslip`, responsável por cadastrar o redirecionamento para a página de pagamento via boleto bancário. Na linha 8, existe uma diretiva de pré-processamento do trecho de código que faz o `log` caso o método seja executado com sucesso e que pertence a característica `Logging` [Gaia *et al.*, 2013].

```
1 public class ControllerServlet extends HttpServlet {
2     public void init() {
3         actions.put("goToHome", new GoToAction("home.jsp"));
4         ///#if defined(BankSlip)
5         actions.put("goToBankSlip",
6                 new GoToAction("bankslip.jsp"));
7         ///#endif
8         ///#if defined(Logging)
9         Logger.getRootLogger().addAppender(new ConsoleAppender(
10             new PatternLayout("[%C{1}] Method %M
11                                 executed with success."));
12         ///#endif
13     }
14 }
```

Figura 3-1 - Exemplo de Código Anotado (Fonte: [Gaia *et al.*, 2013])

A vantagem da Compilação Condicional é o código ser marcado em diferentes granularidades, desde uma linha a um arquivo inteiro [Apel; Kästner, 2009]. No entanto, diretivas de pré-processamento são conhecidas por sua capacidade de "poluir" o código com anotações extras, tornando-o menos legível e mais difícil de entender, de manter e de evoluir, além de introduzir erros de difícil detecção em uma inspeção manual [Spencer, 1992; Bram; Wlfgang; Herman; Ahmed, 2009; Apel; Kästner, 2009].

3.2.1.2. Anotações Visuais

Algumas técnicas podem ser utilizadas para minimizar os problemas apresentados por anotações textuais. Uma dessas técnicas é a de anotações visuais, uma abordagem que limita o poder de expressão de anotações para prevenir erros de sintaxe, sem restringir a aplicabilidade de pré-processadores em problemas práticos [Chen *et al.*, 2005]. Basicamente, essa técnica advoga que anotações devem ser inseridas em trechos de código que possuam valor sintático. Para que funcione corretamente, deve haver um mecanismo

que permita aos desenvolvedores anotarem os elementos dessa estrutura, como classes, métodos ou comandos. Isto requer esforço extra, uma vez que apenas anotações baseadas na estrutura sintática do programa devem ser aceitas [Couto *et al.*, 2011].

A Coloração de Código é uma anotação visual cujo objetivo é anotar o código fonte utilizando cores de fundo diferentes a trechos de código que implementam características distintas [Couto *et al.*, 2011; Oliveira *et al.*, 2010]. Um exemplo é a ferramenta CIDE (*Colored IDE*) [Kästner; Apel, 2008]. Na Figura 3-2, é apresentada a utilização da CIDE na IDE Eclipse. Nessa figura, é mostrado um bloco de código que implementa uma classe denominada `Stack`. O método `push` dessa classe possui trechos de código pertencentes a três características diferentes: Sincronização (vermelho), Persistência (azul) e Logging (verde). Podem ser observados trechos de código compartilhados entre duas ou mais características com uma cor que resulta da junção das cores das características. Isso pode ser visto no trecho em amarelo (vermelho + verde), resultante da interseção das características Sincronização e Logging [Couto *et al.*, 2010].

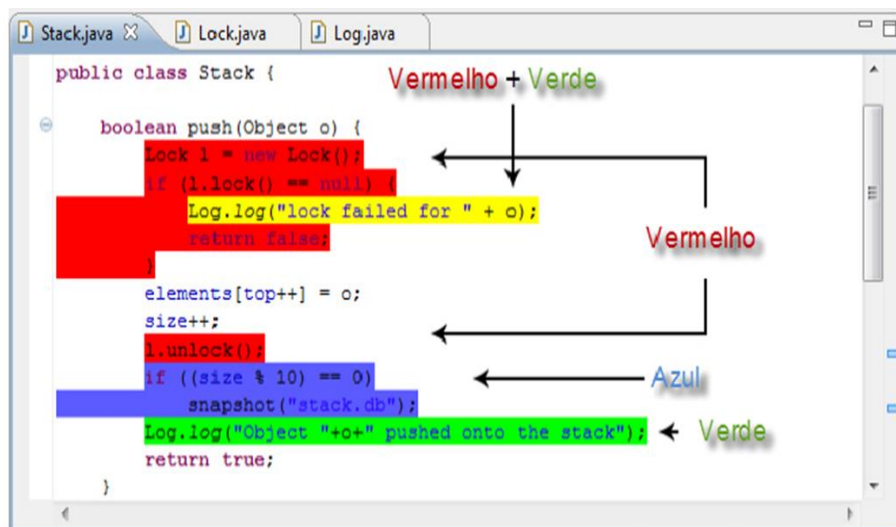


Figura 3-2 - Exemplo de Código Colorido com CIDE (Fonte: [Couto et al., 2011])

Coloração de Código possui a mesma vantagem de Compilação Condicional, o código pode ser marcado em diferentes granularidades, entretanto aumenta a legibilidade, pois não há necessidade de adicionar diretivas de compilação ou quaisquer outras formas de anotações textuais no código fonte. A anotação visual foi criada para resolver problemas decorrentes da anotação textual provendo ferramentas de suporte [Couto *et al.*, 2010].

3.2.2. Baseadas em Composição

As tecnologias baseadas em composições propõem que cada característica seja implementada em um módulo distinto, promovendo a separação física entre o código base e o código da característica. Assim, durante a composição do software, os desenvolvedores devem escolher os módulos de características que devem ser incluídos em determinado produto. Geralmente, esse processo ocorre em tempo de compilação ou em tempo de implantação, o que permite manter o código base separado do código das características [Couto *et al.*, 2010]. Duas tecnologias baseadas em composição têm sido investigadas: orientação a aspectos e orientação a características.

3.2.2.1. Orientação a Aspectos

Programação Orientada a Aspectos (do inglês, *Aspect-Oriented Programming* - AOP) é uma tecnologia proposta para separação de interesses transversais (*crosscutting concerns*) presentes no desenvolvimento de sistemas de software. Interesses transversais são interesses espalhados (*code spreading*) e entrelaçados (*code tangling*) em diversos módulos do software, pois implementam funções e podem afetar diferentes partes do software [Kiczales *et al.*, 1997; Jan, 2000]. Dentre as linguagens com suporte a AOP, AspectJ é considerada a mais madura e estável, sendo uma extensão para linguagem Java com recursos para AOP [Kiczales *et al.*, 1997]. *Code tangling* ocorre quando um interesse transversal encontra-se "misturado" com código responsável pela implementação de interesses não transversais. *Code spreading* ocorre quando um interesse transversal encontra-se implementado em diversas classes.

Na orientação a aspectos, o objetivo é modularizar a implementação de interesses transversais, pois não podem ser adequadamente implementados com a Programação Orientada a Objetos. Isso ocorre por causa de alguns desses interesses violarem a modularização natural do restante da implementação. Para melhor entender como funciona a orientação a aspectos, é necessário ter conhecimento de alguns conceitos básicos dessa tecnologia:

- **Pontos de junção (*join points*)**. São "pontos bem definidos" da execução de um software e definem situações em que há a possibilidade de interceptação do fluxo de execução desse software;

- **Conjuntos de junção (*pointcuts*).** São conjuntos de pontos de junção. Exemplos de pontos de junção são chamadas e execuções de métodos e de construtores, retorno de métodos e de construtores e lançamento e tratamento de exceções;
- **Adendos (*Advices*).** São blocos de código que devem ser executados em um ponto de junção;
- ***Inter-type declarations ou introductions.*** Permitem a introdução de campos e de métodos em classes ou interfaces;
- **Aspecto (*Aspect*).** Caracteriza-se por ser um conjunto de pontos de junção e *advices*, podendo conter ainda *inter-type declarations*.

Na Figura 3-3, é apresentado um trecho de código utilizando AspectJ para o gerenciamento de variabilidade, em que é inserida uma ação de redirecionamento para a página do tipo de pagamento BankSlip. A declaração do `pointcut` nas linhas 2 e 3 é responsável por interceptar o método `init` pertencente a classe `ControllerServlet`. Nas linhas 5 e 6, está declarado um *advice* que insere o comportamento de cadastrar o redirecionamento para a página `bankslip.jsp` após a execução do método interceptado pelo `pointcut` [Gaia *et al.*, 2013].

```

1 public privileged aspect BankSlipAspect {
2     pointcut init(ControllerServlet controller):
3         execution(public void ControllerServlet.init()) && this(controller) &&
4             args();
5     after(ControllerServlet controller): init(controller) {
6         controller.actions.put("goToBankSlip", new GoToAction("bankslip.jsp")
7             );
8     }
9 }

```

Figura 3-3 - Exemplo de Código utilizando AOP (Fonte: [Gaia *et al.*, 2013])

3.2.2.2. Orientação a Características

A Programação Orientada a Características (do inglês, *Feature Oriented Programming* - FOP) é considerada uma tecnologia moderna para modularização e separação de interesses [Gacek; Vukovic, 2000; Gurf *et al.*, 2001]. Foi criada para síntese de programas em LPSs, na qual características são utilizadas para distinguir os sistemas de uma mesma família de produtos [Anastasopoulos *et al.*, 2000]. AHEAD (Algebraic Hierarchical Equations for Application Design) é um conjunto de ferramentas que implementa os conceitos básicos de FOP. Jakarta, seu principal componente, é uma linguagem que possibilita a implementação de refinamentos, ou seja, características em

unidades sistematicamente independentes. Contém também um compilador denominado `composer`, responsável por combinar o código das características com o código base do sistema, e uma linguagem para descrever as combinações válidas de características. Com essa linguagem, o `composer` pode identificar combinações inválidas de características ao tentar gerar um produto na LPS [Couto *et al.*, 2010, Batory, 2006].

FOP tem por base a ideia de que sistemas devem ser sistematicamente construídos por meio da definição e da composição de características, sendo que essas devem ser tratadas como principais abstrações no projeto de sistemas de software. Portanto, devem ser implementadas em unidades de modularização sintaticamente independentes. Além disso, deve ser possível combinar módulos que representam características de forma flexível, sem perder recursos de verificação estática de tipos. Os módulos criados em FOP podem refinar outros módulos de modo incremental, inserindo ou modificando métodos e atributos ou modificando a hierarquia de tipos. Comparando com pacotes em Java, os quais encapsulam um conjunto de classes, refinamentos encapsulam fragmentos de múltiplas classes [Couto *et al.*, 2010].

Na Figura 3-4, é apresentado um refinamento de classe que inclui o comportamento da característica `Logging` na classe. Essa característica foi projetada para registrar a execução com sucesso de métodos públicos [Gaia *et al.*, 2013].

```
1 layer logging;
2 refines class ControllerServlet {
3     public void init() {
4         Super().init();
5         Logger.getRootLogger().addAppender(new ConsoleAppender(
6             new PatternLayout("[%C{1}] Method %M executed with success."));
7     }
8 }
```

Figura 3-4 - Exemplo de Código utilizando FOP (Fonte: [Gaia *et al.*, 2013])

3.2.2.3. Módulos de Características Aspectuais

Módulos de Características Aspectuais (do inglês, ou *Aspectual Feature Modules* - AFM) é uma tecnologia de programação para integrar módulos e aspectos, cujo objetivo é implementar uma relação entre FOP e AOP [Apel *et al.*, 2008; Santos; Valente, 2008; Apel; Batory, 2006]. Os conceitos de AFM estendem a notação de módulo tradicional, pois uma característica é implementada por uma coleção de artefatos, por exemplo, classes,

refinamentos e aspectos [Gaia, 2013]. Um aspecto dentro de AFM não implementa uma função, mas um refinamento por ser mais apropriado. Aspecto é utilizado para modularizar interesses transversais de classes entrelaçados ou espalhados. Assim como as classes e os refinamentos, um aspecto é uma parte integrante de um módulo de características, eles são adicionados e removidos juntamente com a característica a que pertence [Apel; Leich; Saake, 2008; Gaia, 2013].

No geral, AFM refina um sistema de software, utilizando alguma linguagem que implementa refinamentos, por exemplo, Jakarta ou mecanismos orientados a aspectos [Apel; Leich; Saake, 2008]. AFMs independem de linguagens específicas desde que sejam orientadas a aspecto ou orientadas a característica [Apel; Leich; Saake, 2008]. Assim os desenvolvedores podem escolher a técnica que se encaixa melhor a determinado problema, podendo ser aplicadas separada ou concorrentemente.

3.3. Comparação das Técnicas de Extração de LPS

A comparação é baseada nos critérios de caracterização das técnicas abordadas neste trabalho, os quais permitem a observação de semelhanças e de diferenças das técnicas. Os seguintes critérios foram considerados:

1. **Permite coloração em código.** As características são identificadas e coloridas de maneira que o código se torne mais legível. O mesmo mecanismo utilizado em anotação de código, porém utilizando coloração do código fonte;
2. **Suporta criação de módulos.** Ocorre a separação das características do código fonte. Isso é possível com a implementação de alguma metodologia baseada em composição;
3. **Utiliza diretivas de pré-processamento.** Diretivas de pré-processamento são utilizadas para delimitar linhas do código fonte que devem ou não ser incluídas em um determinado produto da LPS;
4. **Permite implementação de características em granularidade grossa.** As características podem estender um programa com código adicional. Esse critério considera que a técnica permite a adição de novas classes ou métodos para o programa ou ampliam pontos de extensão;
5. **Permite implementação de características em granularidade fina.** Os desenvolvedores podem querer introduzir novas declarações sobre métodos existentes e estender expressões ou mesmo assinaturas de método. Esse critério considera que a técnica aplicada possibilita a adição dessas extensões.

A tabela comparativa (Tabela 3-1) contém informações sobre as técnicas apresentadas, sua classificação em relação aos critérios levantados e a justificativa de tal classificação. As técnicas estão posicionadas na vertical e os critérios na horizontal. A comparação recebeu três opções de classificação:

- **Atende:** quando a técnica está de acordo com o critério analisado;
- **Atende parcialmente:** quando a técnica possui particularidades que atendem o critério analisado, porém o critério não é predominante;
- **Não atende:** quando a técnica não está de acordo com o critério analisado.

Tabela 3-1 - Tabela Comparativa

Técnicas Critérios	Compilação Condicional	Coloração de Código	Programação Orientada a Características	Programação Orientada a Aspectos	Módulos de Características Aspectuais
1	Não atende	Atende	Não atende	Não atende	Não atende
	As ferramentas que auxiliam na extração não suportam coloração.	-	Como são extraídos módulos, não tem coloração.	Como são extraídos módulos, não tem coloração.	Módulos são extraídos com ferramentas que não suportam coloração.
2	Não atende	Não atende	Atende	Atende	Atende
	A anotação é feita no próprio código fonte.	As características do código fonte são coloridas sem que haja modularização.	-	-	-
3	Atende	Atende	Não atende	Não atende	Não atende
	-	-	Como módulos são extraídos, não têm diretivas.	Como módulos são extraídos, não têm diretivas.	Como módulos são extraídos, não têm diretivas.
4	Atende	Atende	Atende	Atende	Atende
	-	-	-	-	-
5	Atende	Atende	Atende parcialmente	Atende parcialmente	Atende parcialmente
	-	-	São necessários esforços adicionais para extrair uma característica em granularidade fina.	São necessários esforços adicionais para extrair uma característica em granularidade fina.	São necessários esforços adicionais para extrair uma característica em granularidade fina.

Para fins comparativos, em relação aos critérios de granularidade, AspectJ foi escolhida como representante das diversas técnicas de AOP, assim como Jakarta para FOP. Nessa tabela, pode-se identificar que Compilação Condicional atende três critérios de caracterização. Por ser um mecanismo baseado em anotação textual, atende ao primeiro critério avaliado (**Utiliza diretivas de pré-processamento**). Em relação aos critérios de **granularidade grossa** e **granularidade fina**, compilação condicional se destaca por atender ambos, porém por adicionar diretivas de pré-processamento direto no código fonte o torna menos legível. Coloração de Código possui as mesmas vantagens de Compilação

Condicional; dessa forma, atende aos mesmos critérios que Compilação Condicional, porém esse mecanismo não insere diretivas de pré-processamento diretamente no código, utiliza a coloração de código para aumentar a legibilidade do código.

Em relação a FOP, AOP e AFM, por serem mecanismos baseados em modularização de código, não atendem aos critérios de "**Permite coloração em código**" e "**Utiliza diretivas de pré-processamento**". Porém, durante a extração de uma LPS, foram encontrados diversos problemas, sendo a maioria decorrentes das limitações de AspectJ em lidar com variabilidades de granularidade fina [Kästner; Apel; Batory, 2007]. Por exemplo, características (*features*) que necessitam de extensão em blocos aninhados não podem ser extraídas diretamente para aspectos. Nesses casos, foi necessário inserir métodos `hook` no código para permitir a criação de pontos de junção que possam ser instrumentados por AspectJ. Dessa forma, foi considerado que AOP atende parcialmente ao critério de granularidade fina. Para avaliação de FOP, foi considerado o AHEAD suportar apenas extensões de métodos de granularidade grossa, porém mencionaram a necessidade de reordenar algumas linhas de comando em certas situações para variabilidades de granularidade fina [Liu; Batory; Lengauer, 2006]

Com essa comparação, pode ser observado que as técnicas de extração possuem particularidades. Cabe a equipe responsável pela implantação da abordagem de LPS identificar a técnica que melhor se encaixa ao escopo e aos objetivos da empresa.

3.4. Considerações Finais

Neste capítulo, foram apresentados conceitos relacionados às técnicas utilizadas na extração de LPSs e uma análise comparativa entre elas. Pode-se concluir que cada técnica de extração apresentada neste trabalho possui suas características, variando de anotações textuais e visuais à composição.

As anotações textuais e visuais utilizam diretivas de pré-processamento para extrair as características, porém se diferenciam, pois a primeira realiza anotações no código fonte por meio da utilização de diretivas especiais entendidas pelo pré-processador e a segunda realiza a coloração do código. Essa técnica, apesar de ser relativamente menos complexa de implementar, possui desvantagem em relação as técnicas baseadas em composição, pois não permite que a característica seja separada do programa base. Nas técnicas baseadas em

composição, os trechos de código que implementam uma característica são separados em módulos.

Assim, é importante, para uma empresa de desenvolvimento de software tradicional que queira adotar uma abordagem de desenvolvimento baseada em LPSs, conhecer as técnicas de extração disponíveis, realizar estudos criteriosos e analisar as opções considerando o contexto da empresa.

4. MEDIDAS DE SOFTWARE

4.1. Considerações Iniciais

Medidas de software estão relacionadas à qualidade de software, podendo ser utilizadas para relatar o desempenho atual do sistema e possíveis alterações futuras [Guarizzo, 2008; Gomes, 2013]. Todavia, no contexto deste trabalho, são abordadas medidas de software para avaliar e caracterizar a LPS extraída, TBC-GALL-LPS. Para realizar a avaliação e caracterização da LPS extraída foram consideradas três medidas proposta por Liebig e quatro medidas proposta por Couto [Liebig *et al.*, 2010; Couto, 2010]. Essas medidas de software são apresentadas nas próximas seções.

Três medidas orientadas a tamanho são apresentadas na Seção 5.2. Uma medida de transversalidade é apresentada na Seção 5.3. Uma medida de granularidade é apresentada na Seção 5.4. Duas medidas de localização são apresentadas na Seção 5.5.

4.2. Medidas Orientadas a Tamanho

As medidas orientadas a tamanho são medidas diretas do software, pois atributos são observados, tais como, custo, esforço, quantidade de linhas de código produzidas, quantidade de defeitos registrados e quantidade de linhas de código [Guarizzo, 2008; Gomes, 2013; Couto, 2010]. No contexto deste trabalho, essas medidas destinam-se a avaliar o tamanho dos produtos e das características da LPS implementada com a utilização de Compilação Condicional. Algumas dessas medidas são:

- **Linhas de Código (*Lines of code* - LOC).** Corresponde à quantidade de linhas de código, não contabilizando comentários e linhas em branco;
- **Número de Classes (*Number of classes* - NOC).** Corresponde à quantidade de classes;
- **Linhas de Código de Features (*Lines of Feature Code* - LOF).** Corresponde à quantidade de linhas de código, não contabilizando comentários e linhas em branco, responsável pela implementação de uma dada característica [Liebig *et al.*, 2010].

4.3. Medida de Transversalidade

Medidas de transversalidade destinam-se a medir o comportamento transversal das características extraídas em LPSs baseadas em pré-processadores [Liebig *et al.*, 2010]. A medida utilizada neste trabalho foi **Grau de Espalhamento (*Scattering Degree* - SD).**

Essa medida corresponde à quantidade de ocorrências de constantes `#ifdef` que definem uma característica. Dada uma constante que define uma característica, o SD totaliza a quantidade de ocorrências dessa constante em expressões `#ifdef`.

4.4. Medidas de Granularidade

As características podem estender um programa com código adicional. Extensões com granularidade grossa adicionam novas classes ou métodos para o software. Porém, os desenvolvedores podem querer introduzir novas declarações em métodos existentes, essas são as extensões de grão fino [Kästner *et al.*, 2008]. Medidas de granularidade quantificam o nível hierárquico dos elementos de código anotados para uma característica específica. Nessas medidas, são identificadas características de granularidades grossa e fina. De acordo com a definição, uma característica possui granularidade grossa quando sua implementação ocorre principalmente em unidades sintáticas com um nível hierárquico alto, de acordo com a gramática de Java, tais como, pacotes, classes e interfaces. Por outro lado, uma característica é de granularidade fina quando seu código é composto por unidades sintáticas de baixo nível, tais como, comandos e expressões [Couto, 2010; Kästner *et al.*, 2008]. A medida utilizada neste trabalho foi **Class**. Essa medida corresponde à quantidade de classes inteiramente anotadas para implementar uma característica.

4.5. Medidas de Localização

As medidas de localização fornecem informações sobre a posição dos elementos anotados para implementação das características. O objetivo é mostrar a localização dos comandos associados a cada característica, se eles ocorrem no início ou no final do corpo de um método, antes de um comando *return* ou aninhados a outros comandos [Couto *et al.*, 2011]. Foram utilizadas as seguintes medidas:

- **StartMethod**. Corresponde à quantidade de comandos anotados para uma característica que aparecem no início de um método;
- **EndMethod**. Corresponde à quantidade de comandos anotados para uma característica que aparecem no final de um método.

4.6. Considerações Finais

Neste capítulo, foram apresentadas medidas de software propostas para auxiliar na avaliação e na caracterização de uma LPS. Essas medidas foram utilizadas para avaliar a LPS analisada neste trabalho. Quatro conjuntos de medidas foram abordadas: i) orientadas a tamanho; ii) de transversalidade; iii) de granularidade; e iv) de localização.

5. TRABALHOS RELACIONADOS

Em um dos trabalhos [Bayer *et al.*, 1999], o objetivo foi apresentar a implementação e a análise da LPS da TaRGet. A TaRGeT (*Test and Requirements Generation Tool*) é uma ferramenta de geração de casos de teste, cujo objetivo é lidar com os artefatos de requisitos e de testes de forma integrada, além de gerar casos de teste automaticamente a partir de cenários de casos de uso. Nesse trabalho, foram abordados conceitos sobre Eclipse RCP para o entendimento da implementação da TaRGeT e identificadas características (*features*), aplicando técnicas de extração. Em seguida, foi realizada análise comparativa dos resultados obtidos, destacando vantagens e desvantagens de cada implementação.

Em outro trabalho [Bayer *et al.*, 2000], o objetivo foi estabelecer um processo para extração de LPS a partir de colorações previamente definidas pelas heurísticas baseadas em medidas de modularidade e propagação de mudanças. Nesse trabalho, foi projetada, implementada e avaliada uma ferramenta para auxiliar na extração de LPS para facilitar e orientar o trabalho dos desenvolvedores.

A extração de uma LPS do software ArgoUML, utilizando a Compilação Condicional, foi abordada em outro trabalho [Couto *et al.*, 2011]. O resultado da extração foi avaliado utilizando as medidas de quantidade de linhas de código, de quantidade de pacotes, de quantidade de classes e de quantidade de linhas de código de características (*features*). Além dessas, foram empregadas medidas transversais, medidas de granularidade e medidas de localização.

Em outro trabalho [Almeida, *et al.*, 2005], foi proposta que LPS é uma alternativa para portar produtos existentes para outras plataformas. Além disso, uma abordagem extrativa e incremental foi abordada, utilizando LPS e programação orientada a aspectos. Esse trabalho teve como resultado a identificação de alguns padrões de refatoração por meio de um estudo de caso da indústria. Por fim, a utilidade desses padrões foi avaliada em um estudo de caso da indústria, identificando quais suas limitações de uso.

Os trabalhos mencionados assemelham-se a este no aspecto de lidar com extração de LPSs. No primeiro trabalho [Bayer *et al.*, 1999], são utilizadas técnicas de extração e uma LPS é extraída utilizando cada técnica. Ao final, são realizadas análises sobre as LPSs extraídas comparando os resultados obtidos com a implementação de cada técnica. Este

trabalho foi baseado na Compilação Condicional e foram realizadas análises sobre a LPS extraída. No segundo trabalho [Bayer *et al.*, 2000], o objetivo foi estabelecer um processo para extração de LPS. Neste trabalho, não foi buscada uma nova técnica, mas utilizada uma existente. No terceiro trabalho [Couto *et al.*, 2011], foi realizado um estudo semelhante a este trabalho, sendo uma das principais referências utilizadas para sua realização. Ambos têm o objetivo de extrair uma LPS utilizando Compilação Condicional e realizar medições de acordo com medidas de software. O presente trabalho visa à confirmação dos resultados. No último trabalho [Almeida *et al.*, 2005], é extraída uma LPS, utilizando a abordagem de aspectos com AspectJ, utilizando uma técnica baseada em composição. Ao passo que, a LPS extraída neste trabalho é obtida com uma técnica baseada em anotação textual de código.

6. AVALIAÇÃO

6.1. Considerações Iniciais

Neste capítulo, são apresentados os resultados obtidos com a extração da LPS, utilizando as medidas abordadas no capítulo anterior, e as principais considerações em relação ao processo de extração da LPS. São discutidos e analisados esses resultados para expor as vantagens em adotar a abordagem LPS.

O software utilizado na análise para extrair o TBC-GAAL-LPS é descrito na Seção 6.2. As características identificadas no software analisado são apresentadas na Seção 6.3. O processo de extração é abordado na Seção 6.4. A caracterização da LPS extraída e a análise dos produtos que podem ser gerados a partir dessa LPS são tratadas na Seção 6.5.

6.2. TBC-GAAL

Recomenda-se a aprendizagem da matemática utilizando ferramentas computacionais; com ferramentas exploratórias, é possível uma aproximação dos materiais concretos, ajudando os alunos na construção de raciocínios formais [Lima; Toledo; Costa, 2006]. Uma dessas ferramentas é TBC-GAAL [Lima; Toledo; Costa, 2008], um software educacional para o ensino de assuntos relacionados à Geometria Analítica e Álgebra Linear, desenvolvido na linguagem de programação Java. Nesse software, esses assuntos são abordados em três opções (Figura 6-1): i) Vetores; ii) Cônicas; e iii) Quádricas.

Na opção Vetores (Figura 6-2), há as opções: i) Sistema de Coordenadas 3D; ii) Operações; e iii) Produto Escalar. Na opção Operações, há as opções: i) Adição de Vetores; ii) Subtração de Vetores; e iii) Multiplicação por escalar. Nas duas primeiras opções, pode-se escolher Geometricamente ou Sistemas de Coordenadas 2D e 3D; na última opção, pode-se escolher Sistemas de Coordenadas 2D e 3D. Na opção Cônicas (Figura 6-3), há as opções: i) Elipse; ii) Hipérbole; e iii) Parábola. Na opção Quádricas, há as opções: i) Cilindro, com alternativas Elíptico, Hiperbólico e Parabólico. ii) Elipsóide; iii) Hiperbolóide de 1 folha; iv) Hiperbolóide de 2 folhas; e v) Parabolóide Hiperbólico.

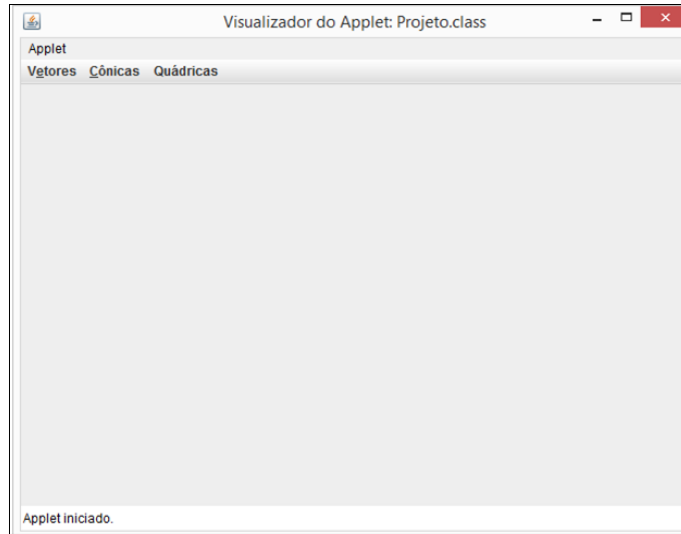


Figura 6-1 - TBC-GAAL: Tela Inicial

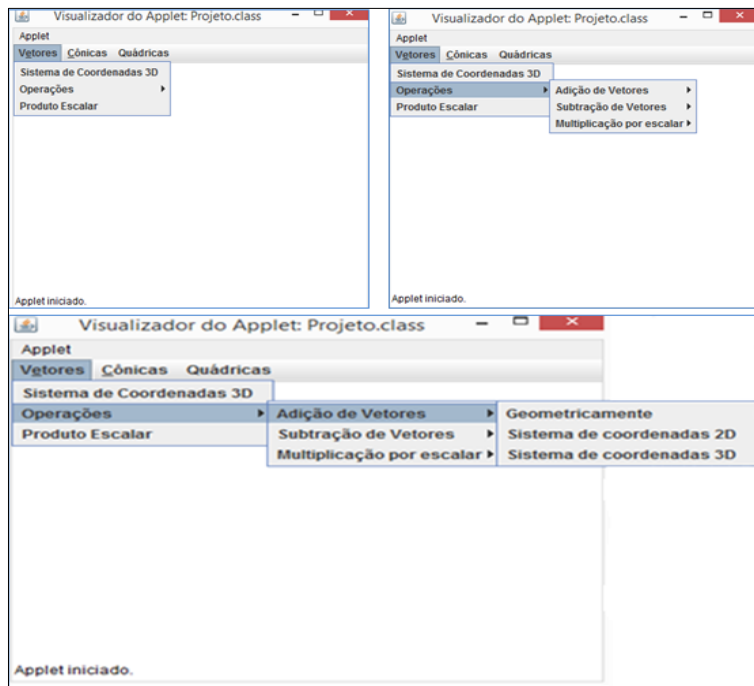


Figura 6-2 - TBC-GAAL: Funções - Vetores

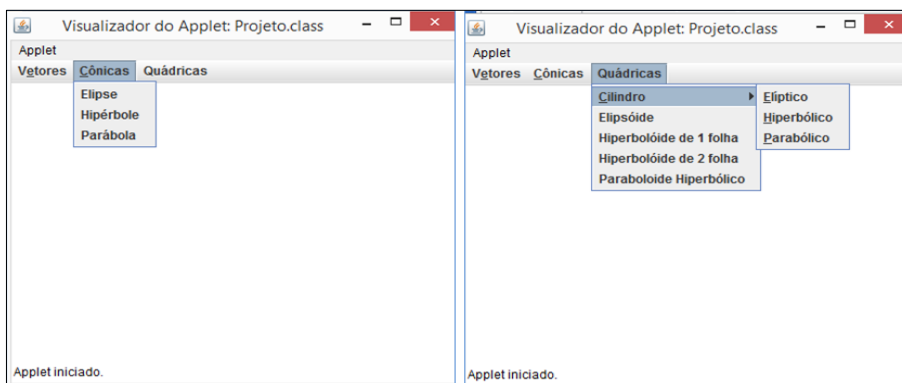


Figura 6-3 - TBC-GAAL: Funções - Cônicas e Quádricas

6.3. Características

Inicialmente, foi realizado um estudo do software TBC-GAAL para entender seu funcionamento e levantar pontos importantes, como quantidade de pacotes e de classes, quais métodos são implementados e como são utilizados, bem como a interação entre as classes e os métodos. Com esse estudo, foi possível identificar classes e declarações não utilizadas, portanto código morto. Esse código foi excluído, pois não agregava valor ao software em termos de funcionalidade, gerando uma variação do TBC-GAAL original, o TBC-GAAL refatorado.

O TBC-GAAL original tinha 50 NOC (quantidade de classes) e 8.811 LOC (linhas de código), as classes `DrawPanelProdutoEscalar`, `PainelProdutoEscalar` e `ProjetoTeste` não eram utilizadas, portanto foram excluídas. Dessa forma, foi obtido o TBC-GAAL refatorado, que possui 47 NOC e 8.211 LOC (aproximadamente 6,8% menor que o TBC-GAAL original). Para a realização da extração da LPS, foi considerado o TBC-GAAL refatorado. A partir do software refatorado, foram selecionadas 10 características, representando requisitos funcionais do software. A característica `OPERAÇÕES` implementa as funções adição e subtração de vetores e a característica `CILINDRO` implementa as figuras elíptica, hiperbólica e parabólica. As demais características são `PRODUTO ESCALAR`, `ELIPSE`, `HIPÉRBOLE`, `PARÁBOLA`, `ELIPSÓIDE`, `HIPERBOLÓIDE DE 1 FOLHA`, `HIPERBOLÓIDE DE 2 FOLHAS` e `PARABOLÓIDE HIPERBÓLICO`, cada uma representando uma função do TBC-GAAL refatorado.

O modelo de características da TBC-GAAL-LPS é mostrado na Figura 6-4. Nesse modelo, foi definida que a característica obrigatória é `Sistemas de Coordenadas 3D`, essa escolha foi feita levando em consideração a relevância das características. As demais características foram consideradas opcionais.

6.4. Processo de Extração

Este trabalho enquadra-se na abordagem extrativa, mas a extração da LPS não será baseada em um conjunto de sistemas e sim de um único sistema. A técnica utilizada para extrair a LPS que originou o TBC-GAAL-LPS foi a de anotação textual utilizando pré-processadores, mais especificamente Compilação Condicional. O ponto de partida para a extração de cada característica foi a identificação das principais classes responsáveis pela

sua implementação. A identificação dos elementos de código fonte do software pertencentes a uma característica foi feita manualmente. Para tal, foi utilizado recursos de busca textual e busca por referência do ambiente de desenvolvimento IDE Eclipse. Nessa tarefa, uma vez identificado que um trecho de código apenas deveria ser compilado caso uma característica fosse selecionada, procedia-se à sua delimitação por meio de `#ifdef` e `#endif`. O código responsável pela implementação das características encontra-se encapsulado em, basicamente, duas classes específicas.

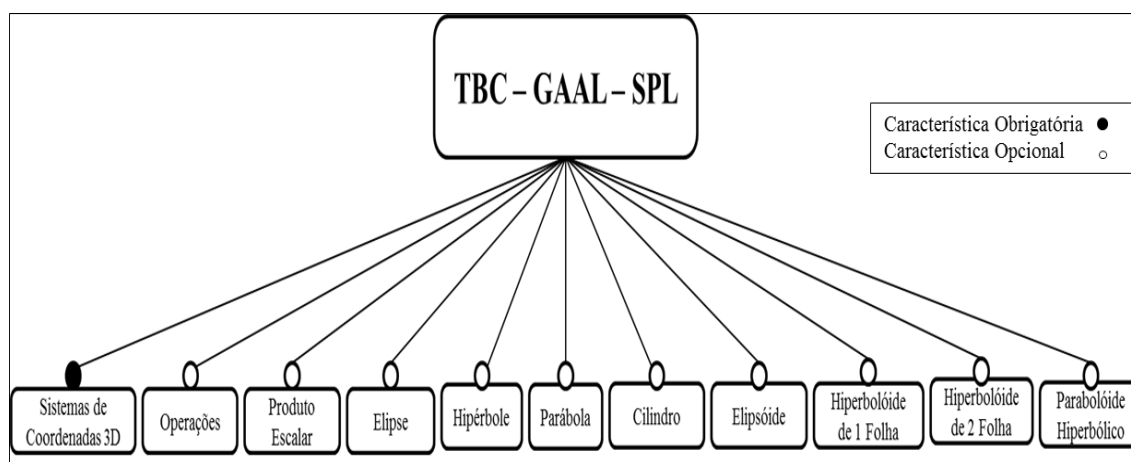


Figura 6-4 - Modelo de Características do TBC-GAAL-LPS

O TBC-GAAL refatorado foi implementado apenas em um pacote; dessa forma, somente as classes foram avaliadas. Um exemplo é a característica PRODUTO ESCALAR, cuja implementação encontra-se nas classes `DrawPanelProdutoEscalar2D` e `PainelProdutoEscalar2D`. Diante dessa característica do código fonte, foi relativamente fácil identificar os trechos de códigos que implementam uma característica, sendo necessário inserir as diretivas de pré-processamento `#ifdef` e `#endif` nas respectivas classes e na classe principal, `Projeto`.

Um segundo exemplo é a implementação da característica OPERAÇÕES que possui uma particularidade, pois tem embutida várias funções. Nesse caso, a implementação dessa característica pode ser encontrada nas classes `DrawPanelAdicaoVetor`, `DrawPanelMultiplicaEscalarVetores`, `DrawPanelSubtracaoVetores`, `DrawPanelSomaVetores`, `DrawPanelMultiplicaEscalarVetores3D`, `DrawPanelSomaVetores3D`, `PainelMultiplicaEscalarVetores3D`, `DrawPanelSubtracaoVetores3D`, `PainelMultiplicaEscalarVetores`, `PainelSomaVetores`, `PainelSomaVetores3D`, `PainelSubtracaoVetores`,

DrawPanelSubtracaoVetor, PainelSubtracaoVetoresGraf, PainelSubtracaoVetores3D e PainelAdicaoVetor. Dessa forma, foi necessário inserir as diretivas de pré-processamento nessas classes e nos trechos de código da classe principal, Projeto, em que essa característica era chamada.

Outro exemplo, similar ao anterior, é a característica CILINDRO cuja implementação pode ser encontrada nas classes DrawPanelElíptico, DrawPanelHiperbolico, DrawPanelParabolico, PainelElíptico, PainelHiperbolico e PainelParabolico. Os dois últimos exemplos podem ser observados na Figura 6-5 em que as linhas 14 a 17 implementam declarações da característica CILINDRO e as linhas 21 a 32 implementam declarações da característica OPERAÇÕES. Como pode ser observado na figura, foi utilizada a diretiva #if defined(), pois é a diretiva reconhecida pelo pré-processador.

```

5 public class Projeto extends JApplet implements ActionListener, ItemListener{
6
7     JPanel panel = new JPanel();
8     private JMenu quadricasMenu = new JMenu( "Quádricas" );
9     private JMenu vetoresMenu = new JMenu( "Vetores" );
10    private JMenu conicasMenu = new JMenu( "Cônicas" );
11    private JMenuItem sistemaCoordenadasItem = new JMenuItem( "Sistema de Coordenadas 3D" );
12
13    // #if defined (CILINDRO)
14    private JMenu cilindroMenu = new JMenu( "Cilindro" );
15    private JMenuItem elipticoItem = new JMenuItem( "Elíptico" );
16    private JMenuItem hiperbolicoItem = new JMenuItem( "Hiperbólico" );
17    private JMenuItem parabolicoItem = new JMenuItem( "Parabólico" );
18    // #endif
19
20    // #if defined (OPERACOES)
21    private JMenu operacoesVetorMenu = new JMenu( "Operações" );
22    private JMenu adicaoVetorMenu = new JMenu( "Adição de Vetores" );
23    private JMenu diferencaVetorMenu = new JMenu( "Subtração de Vetores" );
24    private JMenu multiplicacaoVetorMenu = new JMenu( "Multiplicação por escalar" );
25    private JMenuItem somaVetoresItem = new JMenuItem( "Geomtricamente" );
26    private JMenuItem somaVetoresItem2 = new JMenuItem( "Sistema de coordenadas 2D" );
27    private JMenuItem somaVetoresItem3 = new JMenuItem( "Sistema de coordenadas 3D" );
28    private JMenuItem diferencaVetoresItem0 = new JMenuItem( "Geomtricamente" );
29    private JMenuItem diferencaVetoresItem = new JMenuItem( "Sistema de coordenadas 2D" );
30    private JMenuItem diferencaVetoresItem2 = new JMenuItem( "Sistema de coordenadas 3D" );
31    private JMenuItem multiplicaEscalarVetoresItem = new JMenuItem( "Sistema de coordenadas 2D" );
32    private JMenuItem multiplicaEscalarVetoresItem2 = new JMenuItem( "Sistema de coordenadas 3D" );
33    // #endif

```

Figura 6-5 - Trechos de Código que Implementam as Características CILINDRO e OPERAÇÕES

A avaliação dos produtos gerados pela LPS extraída foi feita por meio de testes funcionais (caixa preta). Esses testes visaram principalmente à validação do produto gerado de acordo com a seleção das características. Em tais testes, foi avaliada a estabilidade do software e comparado o funcionamento do TBC-GAAL refatorado com o produto gerado pela LPS. Por exemplo, considerando um produto com todas as características habilitadas, exceto OPERAÇÕES. Durante os testes, esse produto foi gerado e

executado para avaliar o seu correto funcionamento (se o produto gerado as opções de OPERAÇÕES tinham sido removidas e todas as outras características estavam funcionando corretamente). Esse procedimento foi repetido para todas as características opcionais da LPS extraída.

6.5. Caracterização da LPS Extraída

Para avaliar a LPS extraída, um conjunto de medidas foi utilizado. Por causa das limitações do código fonte, por exemplo, tamanho em termos de linhas de código e as características serem implementadas basicamente em duas classes, foram utilizadas as medidas de tamanho: linhas de código (LOC), número de classes (NOC) e linhas de código de Features (LOF); a medida de transversalidade: Grau de Espalhamento (SD); a medida de granularidade: *Class*; e as medidas de localização: *StartMethod* e *EndMethod*. O valor dessas medidas foi obtido manualmente.

A medida LOF corresponde à quantidade de linhas de código responsáveis pela implementação de uma característica; os comentários e as linhas em branco não são consideradas. Basicamente,

$$\text{LOF (característica)} = \text{LOC (PROD)} - \text{LOC (PROD - característica)},$$

sendo PROD o produto de software com todas as características da LPS habilitadas, PROD - característica um produto de software com todas as características habilitadas, exceto a característica em questão. Na Tabela 6-1, são apresentados os valores coletados para as medidas NOC e LOC. Na Tabela 6-2, são apresentados os valores coletados para a medida LOF e o percentual de código dedicado à implementação de cada característica (considerando o TBC-GAAL refatorado).

Tabela 6-1 - Medidas de Tamanho para os Produtos

Produto	NOC	LOC
GAALApplet - Refatorado	47	8211
Apenas OPERACOES desabilitada	29	4370
Apenas PRODUTO ESCALAR desabilitada	45	7846
Apenas ELIPSE desabilitada	45	7889
Apenas HIPERBOLE desabilitada	45	7879
Apenas PARABOLA desabilitada	45	7850
Apenas CILINDRO desabilitada	41	7056
Apenas ELIPSOIDE desabilitada	45	7938
Apenas HIPERBOLOIDE DE 1 FOLHA desabilitada	45	7915
Apenas HIPERBOLOIDE DE 2 FOLHAS desabilitada	45	7903

Tabela 6-1 - Medidas de Tamanho para os Produtos (cont.)

Produto	NOC	LOC
Apenas PARABOLOIDE HIPERBOLICO desabilitada	45	7949
Todas as features desabilitadas	7	696

Tabela 6-2 - Medidas de Tamanho para as Características

Característica	LOF	
OPERAÇÕES	3841	46,80%
PRODUTO ESCALAR	365	4,45%
ELIPSE	322	3,90%
HIPERBOLE	332	4,04%
PARÁBOLA	361	4,40%
CILINDRO	1155	14,08%
ELIPSÓIDE	273	3,32%
HIPERBOLÓIDE DE 1 FOLHA	296	3,60%
HIPERBOLÓIDE DE 2 FOLHAS	308	3,75%
PARABOLOIDE HIPERBOLICO	262	3,19%
Todas as características	7515	91,52%

Análise do Resultado das Medidas de Tamanho. De acordo com a Tabela 6-1, o menor produto que pode ser derivado a partir da TBC-GAAL-LPS (produto funcional), com todas as características opcionais desabilitadas, possui 696 LOC (aproximadamente 91% menor). Essa redução mostra os benefícios em termos de customização que podem ser alcançados quando se muda para uma abordagem de desenvolvimento baseada em LPS. Analisando a Tabela 6-2, OPERAÇÕES e CILINDRO são as características com maior LOF. De fato, 46,80% e 14,08%, respectivamente, das linhas de código são dedicadas à implementação dessas características. Isso ocorre, pois essas características realizam mais de um cálculo; as outras características contemplam quantidade menor de cálculos. As demais características são relativamente menores e com LOF mais próximo umas das outras (entre 262 LOF e 365 LOF). As características extraídas representam o total de 7.515 LOC (91,52% do total de linhas de código que compõe o TBC-GAAL refatorado). O núcleo da TBC-GAAL-LPS corresponde a aproximadamente 9% do tamanho do software, em termo de linhas de código.

Em relação à medida de transversalidade, grau de espalhamento (SD), que corresponde à quantidade de ocorrências de constantes `#ifdef` que definem uma característica, foram coletados os dados apresentados na Tabela 6-3. Pode-se, novamente, observar que as características OPERAÇÕES e CILINDRO continuam a apresentar maior valor para a medida. Como discutido anteriormente, isso ocorre pois essas características implementam mais de um cálculo matemático. De acordo com a medida de granularidade, *Class*, a quantidade de classes totalmente anotadas para implementar uma característica é

basicamente duas, exceto para OPERAÇÕES e CILINDRO implementadas em 18 e 6 classes, respectivamente. Esse comportamento pode ser observado na Tabela 6-4.

Tabela 6-3 - Medida de Transversalidade: Grau de Espalhamento (SD)

Característica	SD
OPERAÇÕES	21
PRODUTO ESCALAR	5
ELIPSE	5
HIPERBOLE	5
PARÁBOLA	5
CILINDRO	9
ELIPSÓIDE	5
HIPERBOLÓIDE DE 1 FOLHA	5
HIPERBOLÓIDE DE 2 FOLHAS	5
PARABOLÓIDE HIPERBOLICO	5

Tabela 6-4 - Medida de Granularidade: Class

Característica	Class
OPERAÇÕES	18
PRODUTO ESCALAR	2
ELIPSE	2
HIPERBOLE	2
PARÁBOLA	2
CILINDRO	6
ELIPSÓIDE	2
HIPERBOLÓIDE DE 1 FOLHA	2
HIPERBOLÓIDE DE 2 FOLHAS	2
PARABOLÓIDE HIPERBOLICO	2

Para analisar o software em relação às medidas de localização, *StartMethod* e *EndMethod*, foi considerada apenas a classe *Produto*, visto que as demais classes foram quantificadas em relação à medida de granularidade, *Class*, ou seja, estão totalmente anotadas para implementar uma dada característica. Por o software ser implementado de maneira que cada classe seja responsável por uma característica, não foram obtidos resultados interessantes em relação às medidas de localização. Conforme a Tabela 6-5, apenas as características *HIPERBOLE* e *CILINDRO* possuem comandos anotados que aparecem no início e no fim de um método.

Tabela 6-5 - Medida de Localização: StartMethod e EndMethod

Característica	StartMethod	EndMethod
OPERAÇÕES	0	0
PRODUTO ESCALAR	0	0
ELIPSE	0	0
HIPERBOLE	0	1
PARÁBOLA	0	0
CILINDRO	1	0
ELIPSÓIDE	0	0
HIPERBOLÓIDE DE 1 FOLHA	0	0
HIPERBOLÓIDE DE 2 FOLHAS	0	0
PARABOLÓIDE HIPERBOLICO	0	0

A Figura 6-6 e a Figura 6-7 apresentam um trecho de código anotado em que podem ser observados os dados apresentados na Tabela 6-5. Na Figura 6-6, o trecho de código anotado para a característica CILINDRO está localizado no início do método `actionPerformed(ActionEvent e)` e, na Figura 6-7, o trecho de código anotado para característica HIPERBOLE está localizado no fim do mesmo método.

```
75 public void actionPerformed(ActionEvent e){
76     //#if defined (CILINDRO)
77     if ( e.getSource() == elipticoItem ){
78         getContentPane().removeAll();
79         setEnabled( false );
80         setEnabled( true );
81         Paineleliptico painel = new Paineleliptico();
82         getContentPane().add( painel, BorderLayout.CENTER );
83         validate();
84         repaint();
85     }
86     ...
101 }
102 //#endif
```

Figura 6-6 - Exemplo de Anotação do Tipo *StartMethod*

```
...
267     //#if defined (HIPERBOLE)
268     else if ( e.getSource() == hiperboleItem ){
269         getContentPane().removeAll();
270         setEnabled( false );
271         setEnabled( true );
272         Paineleliptico painel = new Paineleliptico();
273         getContentPane().add( painel, BorderLayout.CENTER );
274         validate();
275         repaint();
276     }
277     //#endif
278 }
279 }//metodo actionPerformed( ActionEvent e)
```

Figura 6-7 - Exemplo de Anotação do Tipo *EndMethod*

6.6. Considerações Finais

Neste capítulo, foram apresentadas e discutidas medidas criadas e adaptadas por outros autores [Liebig *et al.*, 2010; Couto, 2010] para avaliar e caracterizar a LPS extraída - TBC-GAAL-LPS. Ao final, pode-se observar a redução de linhas de código obtida com a identificação das características do software TBC-GAAL refatorado (TBC-GAAL-LPS). Foi observado que o menor produto gerado pela LPS é aproximadamente 91% menor, em termos de linhas de código (TBC-GAAL refatorado). É importante ressaltar que esse produto é funcional e essa redução pode contribuir para manutenibilidade do software, visto que um software mais compacto pode ser facilitada a manutenção.

Embora o resultado de algumas medidas tenha sido simples, como das medidas de localização, podem ser percebidos os benefícios que essa abordagem pode trazer, tais como, maior flexibilidade em relação à adição e/ou exclusão de novas características e a possibilidade de ampliar a funcionalidade e o público alvo que o software TBC-GAAL pode atender. Assim, podem ser notadas vantagens a serem alcançadas com a abordagem de LPS, por meio de um estudo prático realizado com o software TCB-GAAL, é perceptível e vantajosa.

7. CONSIDERAÇÕES FINAIS

7.1. Conclusões

O mercado de desenvolvimento de software precisa construir produtos com mais qualidade, redução nos custos, adaptação rápida às mudanças e menor tempo de colocação no mercado atendendo as necessidades dos clientes. Esses objetivos são essenciais para que as empresas possam conseguir vantagem competitiva no mercado. A abordagem de LPSs surgiu como alternativa para alcançar esses objetivos.

No decorrer do desenvolvimento deste trabalho, foi apresentado que, embora LPSs possuam pontos negativos em relação a sua implementação, destacando o custo elevado, retorno em longo prazo e resistência organizacional, contribui de maneira significativa para alcançar esses pontos. Embora o alto investimento inicial, o esforço para manter uma LPS, a partir de um ponto, é menor do que o esforço para manter uma abordagem de desenvolvimento tradicional.

Este trabalho foi realizado com intuito de expor conceitos e técnicas relacionados à LPSs e, principalmente, um exemplo prático de extração de uma LPS. De acordo com os resultados obtidos, pode-se afirmar que LPSs trazem resultados significativos. Conforme exposto na avaliação realizada no software TBC-GAAL-LPS, verificou-se que houve redução, em termos de linhas de código, de aproximadamente 91%, comparado ao TBC-GAAL refatorado.

Além de reduzir o tamanho do TBC-GAAL refatorado, em termos de linhas de código, TBC-GAAL-LPS possui mais flexibilidade em relação a possibilidade de acrescentar e retirar funções do software. Isso é interessante, pois o software pode ser adaptado de acordo com as necessidades dos clientes. Um exemplo é que o TBC-GAAL atende as necessidades da área de Geometria Analítica e Álgebra Linear, mas, futuramente, poder ser estendido abrangendo outras áreas ao adicionar novas características.

Comparando os resultados obtidos neste trabalho com um dos trabalhos relacionados [Couto, 2010], pode-se verificar que ambos tiveram redução no tamanho do software quando gerou o menor produto da LPS extraída, embora o percentual de redução tenha sido maior neste trabalho. Em relação às outras medidas, o trabalho relacionado [Couto, 2010] obteve resultados mais relevantes por causa da estrutura do software analisado.

Pode-se concluir que, com a redução do software TBC-GAAL refatorado, os benefícios em termos de customização que podem ser alcançados quando se muda para uma abordagem de desenvolvimento baseada em LPSs são relevantes. Isto é, LPS extraída permite aos seus usuários criar um sistema do tamanho de suas reais necessidades.

7.2. Contribuições

Foi relatada uma experiência que compreendeu a extração de 10 características de um software real (TBC-GAAL), tendo como finalidade a geração de uma LPS (TBC-GAAL-LPS). As principais contribuições deste trabalho são:

- A extração de uma LPS a partir de um software real;
- Disponibilizar referencial teórico para a comunidade acadêmica e da indústria, mostrando um caso prático e real de extração de LPS.

7.3. Limitações

Foram identificados alguns pontos relevantes que interferiram no resultado final do trabalho desenvolvido. Primeiramente, o tamanho do software escolhido, em termos de linhas de código, limitou a resultados simples para algumas medidas, por exemplo *StartMethod* e *EndMethod*. A linguagem de programação do software, Java, também dificultou o processo de extração, visto que essa não possui suporte para diretivas de pré-processamento. Dessa forma, foi necessária a utilização de uma ferramenta para auxiliar no processo de extração.

7.4. Trabalhos Futuros

Como sugestões de trabalhos futuros, podem incluir a extração de outras características, bem como a automação da LPS extraída. Pode-se também investigar a extração das características do TBC-GAAL utilizando outras técnicas, por exemplo, Programação Orientada a Aspectos, Programação Orientada a Características e Coloração de Código.

REFERÊNCIAS BIBLIOGRÁFICAS

- Ahmed, F.; Campbell, P.; Lagharid, M. S. Cognitive Factors in Software Product Line Engineering. In: International Conference on Computer Modelling and Simulation. pp. 352-355. 2009.
- Almeida, E. S. de; Alvaro, A.; Lucrédio, D.; Garcia, V. C.; Meira, S. R. de L. A Survey on Software Reuse Processes. In: International Conference on Information Reuse and Integration. pp. 66-71. 2005.
- Anastasopoulos, M.; Bayer, J.; Flege, O.; Gacek, C. A Process for Product Line Architecture Creation and Evaluation PuLSE-DSSA - Version 2. In: Fraunhofer IESE Report No. 038.00/E. 2000.
- Apel, S.; Batory, D. When to Use Features and Aspects? A Case Study. In: International Conference on Generative Programming and Component Engineering. pp. 59-68. 2006.
- Apel, S.; Kästner, C. Virtual separation of concerns - a second chance for preprocessors. Journal of Object Technology. 8(6):59-78. 2009.
- Apel, S.; Leich, T.; Saake, G. Aspectual Feature Modules. In: Transactions on Software Engineering. V. 34, I. 2, pp. 162-180. 2008
- Babar, M. A; Chen, L.; Shull, F. Managing variability in software product lines. In: IEEE Software. V. 27. I. 3. pp. 89-91, 2010.
- Bayer, J.; Flege, O.; Knauber, P.; Laqua, R.; Muthig, D.; Schmid, K.; Widen, T.; DeBaud, J. Pulse: A Methodology to Develop Software Product Line. In: Symposium on Software Reusability. pp. 122-131. 1999.
- Bayer, J.; Gacek, C.; Muthig, D.; Widen, T. PuLSE-I: Deriving Instances from a Product Line Infrastructure. In: Engineering of Computer Based Systems. pp. 237-245, 2000.
- Bram A.; Wolfgang, D. M.; Herman T.; Ahmed E. H. Can we refactor conditional compilation into aspects? In: 8th ACM International Conference on Aspect-Oriented Software Development (AOSD). pages 243-254. 2009.
- Carvalho, E. G. de. Globalização e Estratégias Competitivas na Indústria Automobilística: Uma Abodagem a partir das Principais Montadoras Instaladas no Brasil. In: Gestão & Produção. v.12, n.1, pp. 121-133, 2005.
- Chen, Y.; Gannod, G. C.; Collofello, J. S. A Software Product Line Process Simulator. In: International Workshop on Software Process Simulation and Modeling. 2005.
- Clements, P.; Northrop, L. Software Product Lines: Practices and Patterns. Addison-Wesley, 2002.
- Cohen, S. Predicting when Software Product Lines Pays. In: Software Engineering Institute. 2003.

- Couto, M. V. de A. Extração de Linhas de Produtos de Software: Um Estudo de Caso Usando Compilação Condicional. Universidade Católica de Minas Gerais. 61p. 2010.
- Couto, M. V.; Valente, M. T.; Figueiredo, E. Extracting Software Product Lines: A Case Study Using Conditional Compilation. In: European Conference on Software Maintenance and Reengineering. pp. 191-200. 2011.
- Durscki, R. C.; Spinola, M. M.; Burnett, R. C.; Reinehr, S. S. Linhas de Produto de Software: Riscos e Vantagens de sua Implantação. In: Simpósio Internacional de Melhoria de Processos de Software. pp. 155-166. 2004.
- Edson, A. de O. J. x Dissertação do Programa de Pós-Graduação em Ciência da Computação da Universidade Estadual de Maringá. 2005.
- Ferreira, G. C. S. x Dissertação do Programa de Mestrado em Ciência da Computação da Universidade Federal de Uberlândia. 2012.
- Figueiredo, E.; Cacho, N.; Sant'Anna, C.; Monteiro, M.; Kulesza, U.; Garcia, A.; Soares, S.; Ferrari, F.; Khan, S.; Filho, F. C. Evolving Software Product Lines with Aspects: An Empirical Study on Design Stability. In: International Conference on Software Engineering. pp. 261-270, 2008.
- Flege, O.; Knauber, P.; DeBaud, J. PuLSE-DSSA - A Method for the Development of Software Reference Architectures. In: International Workshop on Software Architecture. pp. 25-28, 1998.
- Gacek, C.; Anastasopoulos, M. Implementing Product Line Variabilities. In: Symposium on Software Reusability: Putting Software Reuse in Context. pp. 109-177. 2001.
- Gacek, C.; Vukovic, A. Vital: Representing Software Reference Architectures. In: International Software Architecture Workshop. pp. 105-110, 2000.
- Gaia, F. N. Uma Avaliação Quantitativa de Módulos de Características Aspectuais para Evolução de Linhas de Produto de Software. Dissertação de Mestrado. Universidade Federal de Uberlândia. 62p. 2013.
- Gomes, A. E. Métricas e Estimativas de Software - O Início de um Rally de Regularidade. Disponível em: <<http://www.apinfo.com/artigo44.htm>>. Acesso em: 20/12/2013.
- Guarizzo, K. Métricas de Software. Trabalho de Conclusão do Curso de Ciência da Computação da Faculdade de Jaguariúna. 2008.
- Gurp, J. v.; Bosch, J.; Svahnberg, M. On the Notion of Variability in Software Product Lines. In: Working IEEE/IFIP Conference on Software Architecture. pp.45-54, 2001.
- Henry, F. My Life and Work. The Floating Press. 354p. 2009.
- Heymans, P.; Trigaux, J. C. Software product line: state of the art. Technical report for PLENTY project, Institut d'Informatique FUNDP. Namur. 2003.

- Jan, B. Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach. ACM Press/Addison-Wesley Publishing Co. 354p. 2000.
- Júnior, R. A. de Lima. Comparação entre Ferramentas para Linha de Produtos de Software. Trabalho de Conclusão de Curso. Engenharia da Computação. Escola Politécnica de Pernambuco. 43p. 2008.
- Kang, K. C.; Cohen, S. G.; Hess, J. A.; Novak, W. E.; Peterson, A. S. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical report, Carnegie-Mellon University Software Engineering Institute. 1990.
- Kästner, C.; Apel, S.; Batory, D. A case study implementing features using aspectj. In 11th International Software Product Line Conference (SPLC), pages 223-232. 2007.
- Kästner, C.; Apel, S.; Kuhlemann, M. Granularity in Software Product Lines. In International Conference on Software Engineering. pp. 311-320. 2008.
- Kiczales, G.; Lamping, J.; Mendhekar, A.; Maeda, C.; Lopes, C.; Loingtier, J. M.; Irwin, J. Aspect-oriented programming. In Aksit, M. e Matsuoka, S. (editores), ECOOP'97 _ Object-Oriented Programming, volume 1241 de Lecture Notes in Computer Science, pp. 220_242. Springer Berlin / Heidelberg. 10.1007/BFb0053381. 1997
- Klaus Pohl, Günter Böckle, and Frank J. van der Linden. Software Product Line Engineering: Foundations, Principles and Techniques. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.
- Knodel, J.; Lindvall, M.; Muthing, D.; Naab, M. Static Evaluation of Software Architectures. In: European Conference on Software Maintenance and Reengineering. 2006.
- Liebig, J.; Apel, S.; Lengauer, C.; Kästner, C.; Schulze, M. An Analysis of the Variability in Forty Preprocessor-Based Software Product Lines. In: International Conference on Software Engineering. pp. 105-114. 2010.
- Lima, I. R. ; Toledo, M. C. P. ; Costa, H. A. X. . Aprendizado de Geometria Analítica e Álgebra Linear Utilizando um Software Gráfico via Internet. In: IV Simpósio Brasileiro de Sistemas de Informação (IV SBSI), 2008, Rio de Janeiro. IV Simpósio Brasileiro de Sistemas de Informação (IV SBSI), 2008. v. 1. p. 94-105.
- Lima, I. R. ; Toledo, M. C. P. ; Costa, H. A. X. . Um Software Educacional para o Ensino de Geometria Analítica e Álgebra Linear via Web. In: XXIX CNMAC - XXIX Congresso Nacional de Matemática Aplicada e Computacional, 2006, Campinas/SP. XXIX CNMAC - XXIX Congresso Nacional de Matemática Aplicada e Computacional, 2006.
- Liu, J.; Batory, D.; Lengauer, C.. Feature oriented refactoring of legacy applications. In 28th International Conference on Software Engineering (ICSE). pages 112-121. 2006.

- Matinlassi, M. Comparison of Software Product Line Architecture Design Methods: COPA, FAST, FORM, Kobra and QADA. In: International Conference on Software Engineering. pp. 127-136. 2004.
- Oliveira, V. B. de; Garcia, R.; Valente, M. T. CIDE+: Uma Ferramenta para Extração Semi-Automática de Linhas de Produtos de Software Usando Coloração de Código. Departamento de Ciências da Computação. UFMG. 2010.
- Santos, R. C. dos; Valente, M. T. Uma Comparação Preliminar entre Tecnologias para Implementação de Variabilidades em Jogos para Celulares. Instituto de Informática, PUC Minas. 2008. Disponível em: <http://homepages.dcc.ufmg.br/~mtov/pub/sbgames2008.pdf>. Acesso em: 17/03/2013.
- Schmid K.; Linden, F. J. v. d. and E. Rommes. Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering. Springer-Verlag. 2007.
- SEI - Software Engineering Institute. A framework for software product line practice 4.2. Pittsburgh. Disponível em <<http://www.sei.cmu.edu/plp/framework.html>>. Acesso em: 08 de jul. 2014.
- Silva, F. A. P. da; Neto, P. A. da M. S.; Garcia, V. C.; Muniz, P. F. Linhas de Produtos de Software: Uma Tendência da Indústria. V Escola Regional de Informática. Capítulo 1. pp. 7-31. 2011.
- Spencer H. #ifdef considered harmful, or portability experience with C News. In USENIX Conference. pages 185-197. 1992.