UNIVERSIDADE FEDERAL DO MARANHÃO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

**DIEGO PEREIRA MENDES**

# ENGENHARIA DE DOMÍNIO APLICADA EM LINHA DE PRODUTO DE *SOFTWARE*

São Luís
2012

**DIEGO PEREIRA MENDES**

# ENGENHARIA DE DOMÍNIO APLICADA EM LINHA DE PRODUTO DE *SOFTWARE*

Monografia apresentada ao Curso de Ciência da Computação da Universidade Federal do Maranhão, como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

Orientadora: Prof. Msc. Maria Auxiliadora Freire

São Luís
2012

Mendes, Diego Pereira.

Engenharia de domínio aplicada em linha de produto de software/ Diego Pereira Mendes. – São Luís, 2012.

58f.

Impresso por computador (Fotocópia).

Orientadora: Maria Auxiliadora Freire.

Monografia (Graduação) – Universidade Federal do Maranhão, Curso de Ciência da Computação, 2012.

1.Software – Engenharia 2.Engenharia de domínio 3.Arquitetura reutilizável I. Título.

CDU 004.41

**DIEGO PEREIRA MENDES**

# ENGENHARIA DE DOMÍNIO APLICADA EM LINHA DE PRODUTO DE *SOFTWARE*

Monografia apresentada ao Curso de Ciência da Computação da Universidade Federal do Maranhão, como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

Aprovada em: _____/_____/_____.

BANCA EXAMINADORA

_____

**Profª Maria Auxiliadora Freire** (Orientadora)
Mestre em Ciência de Engenharia
Universidade Federal do Maranhão

_____

**Profº Samyr Beliche Vale**
Doutorado em Informática
Universidade Federal do Maranhão

_____

**Profº Carlos Eduardo Portela Serra de Castro**
Mestre em Informática
Universidade Federal do Maranhão

*Aos meus pais, Antônio Guimarães Mendes Filho e Nilde Pereira Mendes.*
*Aos meus irmãos, Tiago Pereira Mendes e Matheus Felipe.*

## AGRADECIMENTOS

A Deus, que me deu forças e esperança para superar tantos obstáculos e alcançar mais esta meta.

Aos meus pais, Antônio Guimarães Mendes Filho e Nilde Pereira Mendes da Silva, por todo o incentivo e esforço depositados durante todos os anos da minha vida, além de compreenderem os dias festivos que tive que abrir mão.

Aos meus irmãos, Tiago Pereira e Matheus Filipe, pela paciência durante noites de estudo.

À minha namorada Thallyta Adryelle, pelo apoio, compreensão e confiança constantes.

À minha orientadora, Maria Auxiliadora Freire, pelo apoio, receptividade ao tema, orientação e paciência dispensada, mostrando sempre ser prestativa e interessada em ensinar com paciência.

Aos professores do Departamento de Informática pelos seus ensinamentos e aos funcionários do curso, que durante esses anos, contribuíram de algum modo para o meu crescimento pessoal e profissional.

Aos irmãos do Ministério de Louvor Filho de Davi, que estiveram orando e compreendo os dias de estudo e oraram por mim.

Aos amigos de faculdade da turma 2007.2, que sempre estiveram presente ao meu lado em inúmeros trabalhos e provas nesta difícil caminhada de graduação, me ensinando muita coisa.

Ao meu ex-supervisor do PET, Alexandre César Muniz, pela oportunidade dada, onde tal experiência trouxe-me pela primeira vez a experiência de trabalhar na prática com as tecnologias aprendidas no curso.

E a todas as pessoas que acreditaram e torceram por mim em todos esses anos de graduação e no que esta conquista traz de positivo à minha vida.

*"A inovação se distingue entre líder e um seguidor".*
*Steve Jobs*

**RESUMO**

Engenharia de Domínio. Processo que chefia o estabelecimento de uma plataforma (arquitetura) reutilizável e definição de pontos comuns e variáveis de uma Linha de Produto de *Software*. Diferencia-se da Engenharia de Software porque não foca desenvolver um software, mas sim uma arquitetura que será compartilhada por vários softwares. Aperfeiçoa o trabalho cooperativo. Reduz bruscamente o esforço de desenvolvimento de novos *softwares.*

Palavras-chave: Engenharia de Domínio, Linha de Produto de Software, Engenharia de *Software*, Arquitetura reutilizável.

**ABSTRACT**

Domain Engineering. Process that leads the establishment of a platform (architecture) and reusable definition of commonalities and variables of a Software Product Line. It differs from the Software Engineering because it focuses on developing software, but an architecture that is shared by multiple software. Enhances collaborative work. Sharply reduces the effort of developing new software.

Keywords: Domain Engineering, Software Product Line Engineering, Software Architecture reusable.

## LISTA DE FIGURAS

## LISTA DE TABELAS

## LISTA DE SIGLAS

ELPS - Engenharia de Linha de Produtos de Software

FMP - *Feature Modeling Plug-in*

LPS - Linha de Produto de Software

LP - Linha de Produto

IDE - *Integrated Development Environment*

API - *Application Programming Interface*

**SUMÁRIO**

# 1 INTRODUÇÃO

Na perspectiva de desenvolvimento para reuso, deve ser ressaltada a importância de se pensar de forma mais abrangente sobre o conceito de *produto reutilizável*, ou seja, em vez de ser explorada apenas em nível de código, a reutilização deve acontecer através de artefatos situados em níveis mais altos de abstração, e que, portanto, permitam o reuso de itens como soluções de projeto, infraestrutura de domínio e, idealmente, estruturas de conhecimento, para que possam ser compartilhada por uma família de produtos.

Desse modo, como processo de desenvolvimento desses artefatos, é adotada a disciplina de ***Engenharia de Domínio.***

Para (GUIZZARDI, 2000)*:*

> [...] uma espécie de engenharia de software que atua em um meta-nível, capaz de desenvolver infraestruturas reutilizáveis, que capturam a estrutura e o conhecimento de uma família de aplicações [...]

A Engenharia de Domínio pode ser dividida nas seguintes fases: **gestão do produto, engenharia de requisitos de domínio, desenho do domínio e a realização do domínio.**

Com o uso adequado da Engenharia de Domínio pode-se falar sobre uma nova forma de desenvolvimento: A Linha de Produto de Software. Essa nova forma de desenvolvimento se baseia em dois princípios fundamentais: customização em massa e plataforma. A customização em massa se caracteriza pela produção em larga escala de bens adaptados de acordo com as necessidades individuais do usuário, já a plataforma é qualquer base de tecnologia sobre a qual outras tecnologias ou processos são construídos.

No decorrer deste trabalho será explicado como é possível juntar as características da Engenharia de Domínio no desenvolvimento de uma Linha de Produto de Software.

## 1.1 Justificativa do tema

A proposta da realização deste trabalho monográfico de conclusão de curso é apresentar o uso Engenharia de Domínio em LPS à comunidade acadêmica,

de modo que esta boa prática seja mais difundida entre os profissionais da informática.

O uso da Engenharia de Domínio emprega-se na resolução de um dos problemas mais enfrentados pelas empresas de *software* no mercado, desenvolver um grande número de *softwares* para atender a demanda do mercado, sem ter muito gasto e com produção em velocidade acelerada.

Outro problema bastante enfrentado por quem desenvolve sistemas é o esforço que muitas das vezes é realizado e não é reaproveitado.

Para resolver tais problemas, surgiu o conceito de Engenharia de Domínio, que visa à construção de uma arquitetura que será reutilizada por vários *softwares* que estão dentro de um mesmo domínio (porção do mercado com características parecidas) criando assim uma Linha de Produto de Software.

## 1.2 Objetivos da pesquisa

### 1.2.1 Geral

Estudar o uso da Engenharia de Domínio na criação de uma Linha de Produto de Software.

### 1.2.2 Específico

a) Apresentar a Engenharia de Domínio em LPS e a teoria que a cerca;

b) Expor os processos para a realização de uma Engenharia de Domínio e os conceitos que este utiliza na sua abordagem;

c) Apresentar informações relevantes sobre a construção de uma LPS;

d) Mostrar uma aplicação utilizando a LPS.

## 1.3 Organização do trabalho

Esta monografia encontra-se organizada em capítulos. Será apresentado no capítulo 1 a introdução necessária ao restante dos capítulos que compõem o trabalho.

No capítulo 2, mostra-se a fundamentação teórica da pesquisa. De início, apresentam-se os conceitos sobre a Engenharia de Domínio e como seus passos de desenvolvimento estão divididos.

No capítulo 3, serão apresentado os conceitos que formam a Linha de Produto de Software, e seus 3 passos para a criação de uma LPS,

No capítulo 4, exibe-se o estudo de caso onde o desenvolvimento de um sistema de calculadora é guiado pelas técnicas da Engenharia de Domínio e LPS.

Por fim, apresentam-se no Capítulo 5 as considerações finais e o futuro referente a esta pesquisa.

# 2 ENGENHARIA DE DOMÍNIO

A Engenharia de domínio é o processo sistemático de Análise, Projeto e Realização/Implementação de componentes reutilizáveis de Software de um domínio executado independente de um projeto concreto, ou em outras palavras conforme Bacelo:

> A Engenharia de Domínio é quem constrói modelos de domínio com o apoio oferecido por seus processos. Processos de Engenharia de Domínio, geralmente, compreendem as fases de análise, projeto e implementação. Na fase de análise, é feito o levantamento dos requisitos do domínio, normalmente especificados através de modelos de características. Na etapa de projeto, modelos de análise são refinados visando à construção de uma arquitetura do domínio, conhecida como Arquitetura de Software Específica para Domínios ou como DSSA (***Domain Specific Software Architecture***). Por último, a fase de implementação visa à geração de código fonte para a arquitetura gerada no projeto do domínio. (BACELO, 2009)

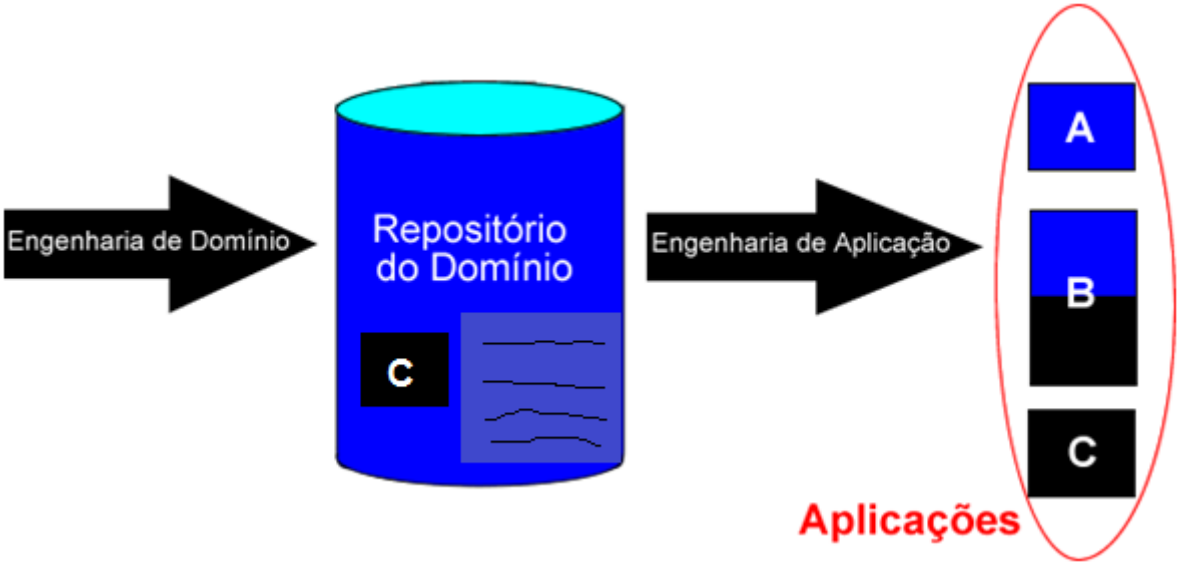


**Figura 2-1: Engenharia de Domínio x Engenharia da Aplicação**

Observe na Figura 2-1 que a Engenharia de Domínio cria por meio dos seus processos o núcleo "**C**", um arquivo de configuração e os artefatos reutilizáveis que unidos poderão formar diferentes aplicações em um determinado domínio. Pode-se ainda resumir dizendo que a Engenharia de Domínio se preocupa com o desenvolvimento de uma arquitetura reutilizável em vez de um *software* único, o objetivo dela é por meio de seus processos, desenvolver uma arquitetura que servirá de alicerce, base, para que um conjunto de *softwares* compartilhem sua implementação.

## 2.1 Engenharia de domínio x Engenharia de aplicação

O modelo de desenvolvimento de uma Linha de Produto de *Software* encontra-se dividida em dois processos importantes: A Engenharia de Domínio e a Engenharia de Aplicação.

- **A Engenharia de Domínio** é o processo que chefia o estabelecimento de uma plataforma (arquitetura) reutilizável e definição de pontos comuns e variáveis da LPS. A plataforma é a reunião de todos os tipos de artefatos de software (requisitos, desenho, realização e testes) sendo que a ligação entre estes artefatos facilita a reutilização sistemática e consistente. A Engenharia de Domínio pode ser comparada com uma Engenharia de Software conforme a Tabela 2-1:

**Tabela 2-1: Comparação entre a Engenharia de Software e Engenharia de Domínio**

| Engenharia de Software | Engenharia de Domínio |
|---|---|
| Análise de requisitos | Análise de domínio |
| Especificação do sistema | Especificação da infraestrutura |
| Projeto e implementação | Implementação da infraestrutura |

O próprio nome Engenharia de Software já traz o conceito de criação, análise, desenvolvimento e manutenção, seguindo-se os passos de análise de requisitos obtêm-se a especificação de um sistema único, que será projetado e implementado para solucionar uma instância de um problema existente em um domínio, em outras palavras, imagine um domínio como campo da matemática onde professores e demais profissionais necessitam de softwares que auxiliem os mesmo em suas atividades, pode ocorrer que um profissional exija o desenvolvimento de um *software* que permita realizar contas simples (somar, subtrair, multiplicar e dividir), ao seguir os passos da engenharia de software e implementar este sistema, será verificado que o *software* desenvolvido apenas resolverá um conjunto de problemas no domínio da matemática, o sistema apenas soma, subtrai, multiplica e divide, ele não foi desenvolvido para atender as outras demandas de problemas, não foi pensado como uma linha de produto, sendo assim, para resolver os outros problemas, deverá ser criado um novo *software*, em vez de apenas receber pequenas adaptações, que é uma das vantagens da LPS. Nesse contexto é importante que seja feito uma Engenharia de Domínio, pois esta sempre imagina e

pensa em família de aplicações após uma análise de domínio (ex: verificar os processos que são mais usados, como funciona o trabalho dos profissionais matemáticos, análise de alguns sistemas que já são usados). Não seria desenvolvido um software único, mas uma arquitetura que seria reutilizada para desenvolver vários sistemas, ou melhor, preparado para desenvolver uma família de sistemas. No exemplo citado, a arquitetura se resumiria em apenas fazer cálculos, as operações matemáticas seriam componentes prontos para serem adaptados e mesclados com a arquitetura para formar softwares capazes de atender muitas instancias de problema do domínio, observe na Figura 2-2.

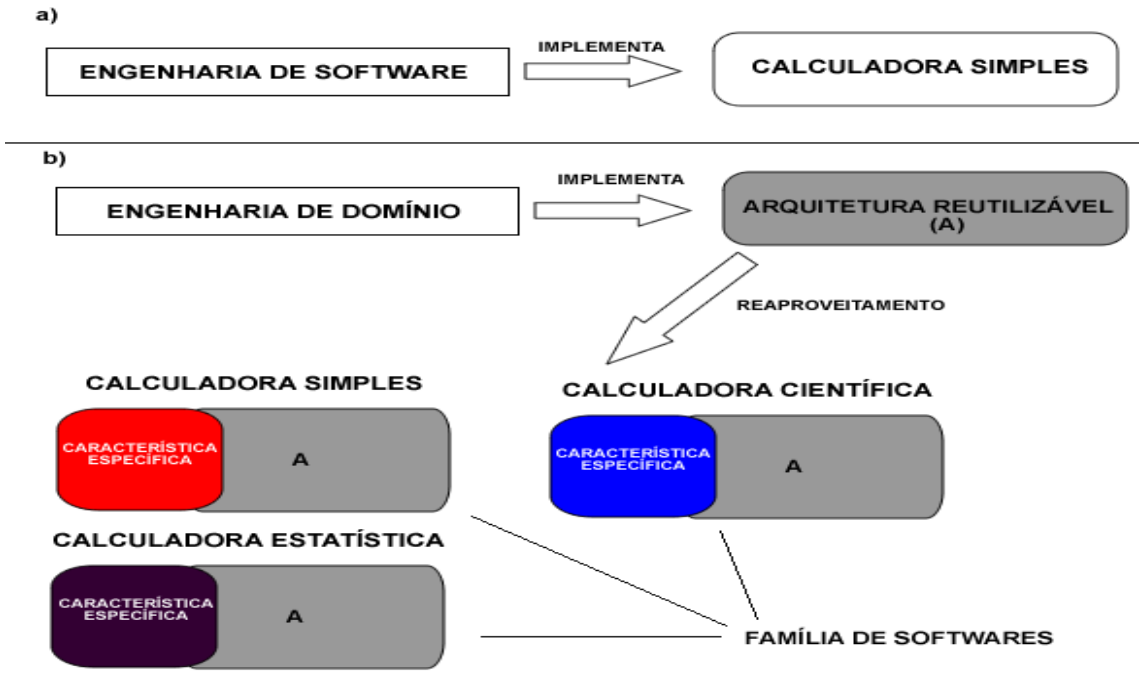


**Figura 2-2: Diferença entre Engenharia de *Software* e Engenharia de Domínio**

Na parte “a” da Figura 2-2 a Engenharia de *Software* ao longo da sua execução tem em vista programar um sistema único, enquanto na parte “b” observa-se que a Engenharia de Domínio visa programar uma arquitetura que será reutilizada e compartilhada por uma família de sistemas, onde cada sistema da família, como mostrado na Figura 2-2 com as cores vermelha, azul e roxo, possuirá especialidades que os diferenciam um dos outros.

- **A Engenharia de Aplicação** é o processo que chefia a derivação das aplicações da LPS, a partir da plataforma estabelecida na engenharia de domínio, e

a exploração da variabilidade da LPS, assegurando a ligação correta das variabilidades de acordo com as necessidades específicas da aplicação. A Engenharia de Domínio e a Engenharia de Aplicação são complementares, trabalhando em processos paralelos que formam um modelo base, ou seja, um sistema de produção de software orientado à reutilização. Um processo de Engenharia de Aplicação desenvolve produtos de software, a partir de artefatos de software criados pela engenharia de domínio.

A Engenharia de Domínio simboliza o princípio do desenho para reutilização, enquanto a Engenharia de Aplicação simboliza o princípio do desenho com reutilização.

O Processo de Engenharia de Linha de Produtos de Software (PELPS) é um modelo de processo de software que evolui, por meio de várias iterações. Assim os sistemas desenvolvidos são capazes de se adaptar a mudanças nos requisitos durante cada iteração de desenvolvimento.

## 2.2 Processos da engenharia de domínio

Durante a fase de Engenharia de Domínio as semelhanças e variabilidades nas linhas de produtos são analisadas do ponto de vista geral dos requisitos. Esta fase é composta por cinco processos, sendo: gestão do produto, engenharia de requisitos de domínio, desenho de domínio, realização de domínio e testes de domínio.

Pode-se verificar de uma forma gráfica na Figura 2-3 o fluxo dos processos da Engenharia de Domínio.

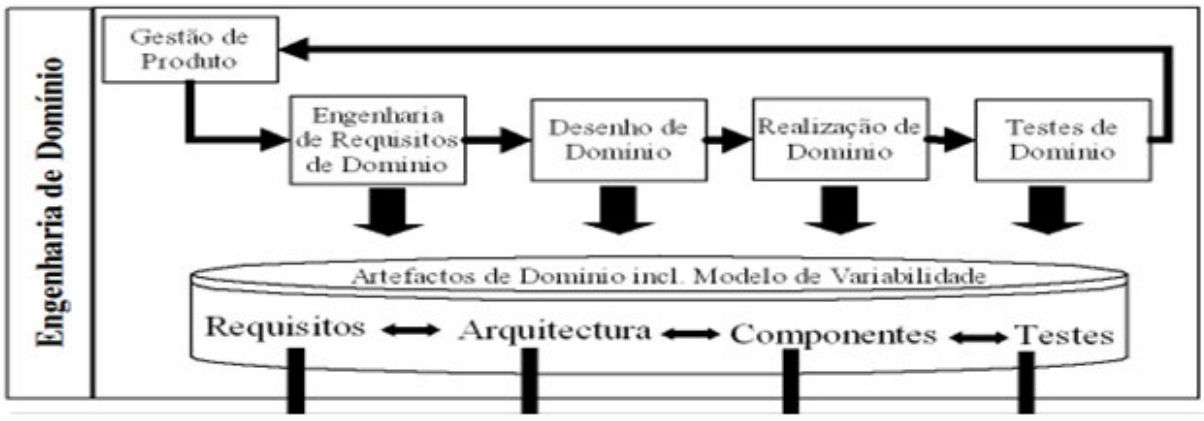


**Figura 2-3: Processos da engenharia de domínio (CLEMENTS, 2002)**

### 2.2.1 Gestão do produto

Este processo permite lidar com os aspectos econômicos de uma LPS e em particular com a estratégia de mercado. A principal função é a gestão da lista de documentos de um produto da companhia ou unidade de negócio. A ELPS permite definir o que está dentro da área da linha de produtos e o que está fora.

### 2.2.2 Engenharia de requisitos de domínio

Segundo IAN (IAN, 2004) as atividades principais do processo de engenharia de requisitos são as seguintes: estudo de viabilidade, levantamento e análise de requisitos, especificação de requisitos e validação de requisitos.

O estudo de viabilidade corresponde a uma descrição resumida do sistema e da forma como deverá ser utilizado dentro da organização. Este estudo tem por objetivo recolher e avaliar a informação necessária para garantir uma solução viável para o sistema quanto à tecnologia, custos, tempo e enquadramento organizacional.

Engloba todas as atividades para obter e documentar os requisitos comuns e variáveis da linha de produtos. A entrada deste processo é o roteiro do produto adquirido na fase de gestão do produto. A saída deste processo compreende os requisitos textuais reutilizáveis, em particular, o modelo de variabilidade da linha de produto. Assim, a saída não inclui a especificação de requisitos de um determinado aplicativo, mas as exigências comuns e variáveis para todas as aplicações previsíveis da linha de produtos, esta fase engloba todas as atividades para eliciar e documentar os requisitos comuns e variáveis da linha de produtos.

As atividades para levantamento e análise de requisitos incluem:

- Compreensão do domínio: onde o analista deve fazer um estudo do domínio da aplicação;

- Levantamento de necessidades dos *stakeholders*: o analista interage com os *stakeholders* [1] do sistema para identificar as suas necessidades e averiguar aquilo que deve ser construído;
- Classificação de requisitos: os requisitos depois de obtidos são organizados em grupos funcionais;
- Resolução de conflitos: quando existem muitos *stakeholders* envolvidos aumenta a possibilidade de conflitos nos requisitos, pelo que importa resolvê-los;
- Estabelecimento de prioridades nos requisitos: esta fase compreende a interação com os *stakeholders* para descobrir os requisitos mais importantes;
- Verificação de requisitos: serve para descobrir se estão completos, se são consistentes e de acordo com o que realmente desejam os *stakeholders* do sistema.

### 2.2.3 Desenho de domínio

Engloba todas as atividades para definir a arquitetura de referência da linha de produto. A arquitetura de referência fornece uma estrutura comum para todos os aplicativos da linha de produto. A entrada para este processo consiste nos requisitos de domínio e o modelo de variabilidade da engenharia de requisitos de domínio. A saída engloba a arquitetura de referência e um modelo de variabilidade.

#### 2.2.3.1 Elaboração do modelo de caso de uso

Com os requisitos da fase anterior será gerado o modelo de caso de uso, pois é ponto de partida para a identificação das *features* [2]e, como consequência das variabilidades de uma LP.

Durante a elaboração do modelo de caso de uso, será necessário expressar a variabilidade de requisitos da Linha de Produto de Software em **Diagrama Caso de Uso,** e também na descrição dos cenários do diagrama de caso de uso, pode-se adotar os modelos demonstrados nos ANEXO A e ANEXO B respectivamente.

---

[1] *Stakeholders é um usuário interessado em um projeto*
[2] *Features é uma qualidade ou característica de um sistema que é notável ao usuário*

#### 2.2.3.2 *Configuration knowledge*

É um arquivo que definirá quais agrupamentos de *features* são válidas para gerar um novo produto, ele é importante, pois contém uma lista de itens de configuração em que expressões de *features* e tarefas (transformações) são usadas para geração automática de membros da LPS.

Portanto, o ***configuration knowledge*** serve como um guia no processo de montagem dos membros da família de produtos de software. Ele determinará quais os tipos de produtos são possíveis com a arquitetura e os componentes criados.

#### 2.2.3.3 Modelo de *features*

Esta atividade visa construir o modelo de *features* da LP a partir do modelo de casos de uso. A elaboração do modelo de *features* segue as etapas de identificação das *features*, definição do tipo das *features* e representação gráfica das *features*.

### 2.2.4 Realização de domínio

Esta fase se preocupa com o detalhe do desenho e a implementação de artefatos de software reutilizáveis (componentes). A entrada para este processo consiste no modelo de variabilidade e a arquitetura de referência (não codificada) que foi implementada no processo anterior (Desenho de Domínio), incluindo uma lista de artefatos de software reutilizáveis. Nesta fase todos serão codificados.

A saída da realização de domínio compreende o projeto detalhado e a implementação dos componentes de software que serão reutilizáveis.

Um dos processos definidos anteriormente que merece destaque é Engenharia de Requisitos de Domínio, visto que possuindo tais requisitos pode-se definir um desenho do domínio e a realização do domínio.

### 2.2.5 Testes de domínio

Esta fase se preocupa apenas em verificar se a LPS gerada atendeu as expectativas esperadas, verificando se obedeceu corretamente o arquivo de configuração da LPS e corrigindo pequenos erros não esperados.

# 3 LINHA DE PRODUTOS DE SOFTWARE (LPS)

De acordo com a *Software Engineering Institute* (SEI, 2007) “uma linha de produto de software (LPS) é uma família de sistemas que possuem um conjunto de ativos em comum, possuindo características que satisfaçam as necessidades específicas de um determinado segmento de mercado ou missão e que são desenvolvidos a partir de um conjunto comum de bens essenciais de uma forma preestabelecida”.

## 3.1 Conceitos básicos da LPS

Dentro desta definição, alguns termos devem ser destacados, pois representam as características mais importantes de uma linha de produto e sustentam as principais vantagens propostas pelo modelo. Os termos são: conjunto de ativos em comum, forma preestabelecida e segmento particular de mercado ou missão. A seguir, esses termos serão descritos com base em (CLEMENTS, 2002):

- **Conjunto de ativos em comum**: Os ativos são a essência da linha de produto, correspondem a um conjunto de elementos que podem ser alterados ou personalizados, utilizados na construção dos softwares produzidos (produtos). Entre os ativos estão, por exemplo: componentes de software, modelos de documentos utilizados no processo, ***design-patterns***[3] utilizados pela equipe de desenvolvimento, documentação dos requisitos comuns à família de produtos, a arquitetura da linha de produtos (que será à base da arquitetura de cada produto gerado), cronogramas etc. Dentre estes elementos, a arquitetura é o elemento chave e normalmente é estudada a parte dos outros ativos.

- **Forma preestabelecida**: O processo de produção de softwares através de uma linha de produto é realizado através de planos de produção. Esses planos são definidos para cada ***software*** (produto) que será produzido pela linha. Ao definir um plano de produção, que dará origem a um novo produto, devem-se relacionar quais ativos farão parte deste produto e assim vincular aos processos os anexos de cada ativo utilizado. Os anexos são pequenos processos de utilização contidos em cada ativo e que definem o que o ativo faz, qual a sua flexibilidade, qual a técnica de configuração do ativo. Essa natureza preestabelecida de funcionamento

[3] *Design-patterns é o mesmo que* padrões de desenvolvimento

do processo produtivo da linha de produto é que garante o ganho de tempo e confiabilidade no desenvolvimento dos produtos.

- **Segmento particular ou missão**: Muitas vezes chamado de domínio, refere-se ao corpo de conhecimento ou à área de especialização em que a linha de produto atua. O domínio está diretamente relacionado com o conjunto de funcionalidades correlacionadas que os produtos da linha pretendem atender. Como a flexibilidade dos ativos (especialmente da arquitetura da linha) é limitada, o modelo exige uma delimitação de um segmento de atuação. Sem essa delimitação o escopo da linha de produto poderia ser muito abrangente o que tornaria muito custoso criar e manter o conjunto comum de ativos.

O *Software Engineering Institute* (SEI, 2007) afirma que a linha de produto pode ajudar as organizações a superar os problemas causados pela escassez de recursos. Organizações de todos os tipos e tamanhos descobriram que uma estratégia de linha de produto quando habilmente implementada pode produzir benefícios, e finalmente dar para as organizações uma vantagem competitiva.

Linha de produtos de *software* refere-se a especialidades da engenharia para a criação de um conjunto de sistemas de *softwares* homogêneos, a partir de um conjunto compartilhado de ativos de *software*, utilizando um meio comum de produção. Os fabricantes têm usado por muito tempo as técnicas de engenharia idênticas para criar uma linha de produtos similares usando uma fábrica comum, que monta e configura elementos concebidos para ser reutilizado em todos os produtos diferentes da linha de produtos. Por exemplo, os fabricantes de automóveis podem agora criar muitas variações exclusivas de um modelo de carro usando um único conjunto de peças cuidadosamente arquitetado e uma fábrica projetada especificamente para configurar e montar essas peças.

A ideia de software na fabricação de peças reutilizáveis tem sido usada há décadas, mas o sucesso tem sido difícil. Avanços recentes no campo da linha de produtos de software têm demonstrado que a aplicação estreita e estratégica destes conceitos pode gerar grandes melhorias no tempo de colocação no mercado, ou ***time-to-market***[4], qualidade e custo de engenharia de software. O resultado é muitas vezes um salto descontínuo em vantagem comercial competitiva, semelhante à observada quando os fabricantes adotam a produção em massa e paradigmas de personalização em massa, ou seja, a característica que distingue as linhas de

[4] *Time-to-market é o tempo necessário que o software leva para estar disponível no mercado*

produtos de software a partir de esforços anteriores (reutilização habitual) é a reutilização de software preditivo em relação ao oportunista. Ao invés de colocar componentes de software em uma biblioteca na esperança de que oportunidades de reuso irão surgir, as linhas de produtos de software apenas chamam para artefatos de software (*core assets*[5]) a ser criado quando a reutilização é previsto em um ou mais produtos de uma linha de produtos bem definidos, ou seja, uma LPS visa o desenvolvimento em larga escala, onde produtos são obtidos a partir do reuso de *core assets* previamente construídos e disponíveis em um repositório, na forma de componentes, padrões ou *frameworks*. O *core assets* da LPS é composto por funcionalidades comuns. As características (funcionalidades) comuns constituem o kernel da LPS e são encontradas em todos os produtos da LPS. Como mostrado na Figura 3-1.

Em uma LPS, é feito um estudo de quais características comuns serão reutilizadas entre os produtos que irão ser propagados, (diferente do que acontece em sistemas únicos (reutilização habitual), nos quais o reuso não é planejado, é oportunista, como quando um simples trecho de código de um produto existente é reutilizado em um novo produto) e, a partir desse planejamento, são construídos os artefatos comuns (*core assets*) (1 na Figura 3-1) que, posteriormente, são utilizados para gerar produtos variados da linha (2, 3 e 4 na Figura 3-1). Como mostrado na Figura 3-1, nenhum dos produtos gerados é igual ao outro, cada um deles possui variações específicas.

Krueguer em (KRUEGER, 2011) afirma que linha de produtos de software pode ser descrita em termos de quatro conceitos simples:

- **Software de entradas de ativos**: uma coleção de ativos de software, tais como os requisitos, código-fonte dos componentes, casos de teste, arquitetura e documentação que pode ser configurado e composto de diferentes maneiras para criar todos os produtos em uma linha de produtos. Cada um dos ativos tem um papel bem definido dentro de uma arquitetura comum para a linha de produto. Para controlar variação entre os produtos, alguns dos ativos podem ser opcionais e alguns dos ativos podem ter pontos de variação internos que podem ser configurados de diferentes maneiras para fornecer um comportamento diferente.

[5] *Core Assets* são os artefatos que podem ser reaproveitados por outros produtos da LPS

- **Modelo de decisão e decisões sobre o produto:** O modelo de decisão descreve os recursos opcionais e variáveis para os produtos da linha de produtos.
- **Mecanismo de produção e processo:** os meios para a composição e configuração dos produtos a partir das entradas de ativos de software. Decisões de produto são usadas durante a produção para determinar quais entradas de ativos de software usar e como configurar os pontos de variação dentro desses ativos.
- **Saídas de produtos de software**: a coleção de todos os produtos que podem ser produzidos para a linha de produtos. O âmbito da linha de produtos é determinado pelo conjunto de saídas de produtos de software que podem ser produzidos a partir dos ativos de software e modelo de decisão.

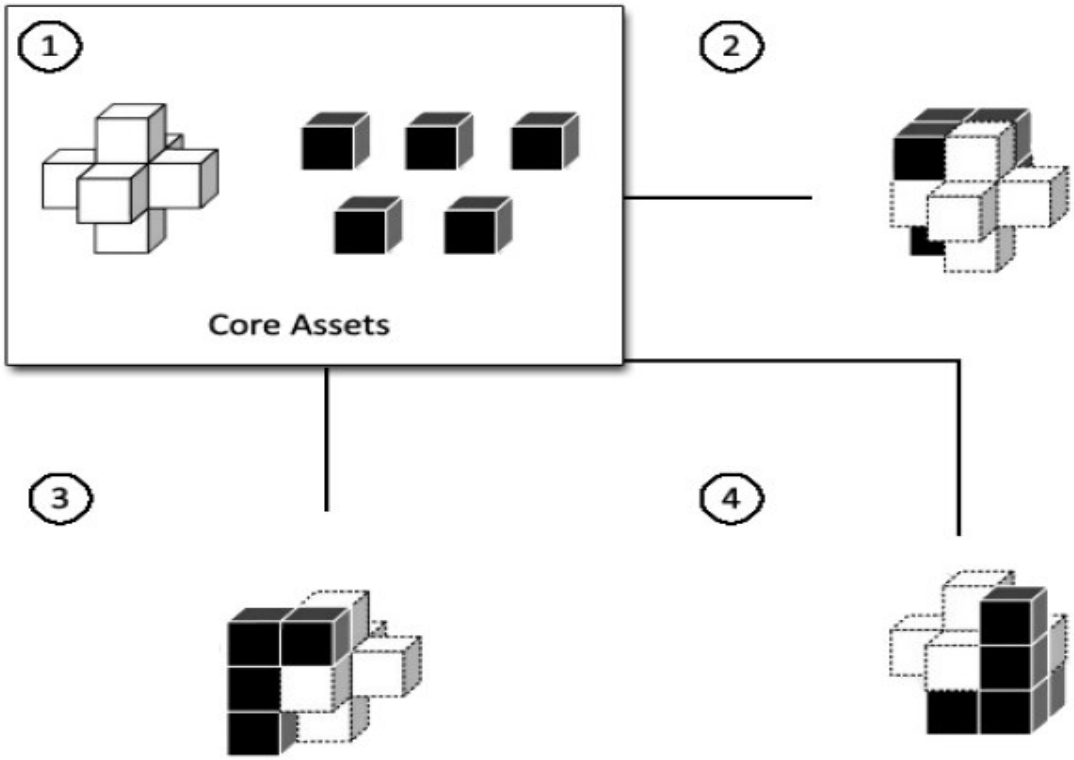


**Figura 3-1: (1) Core Assets / (2, 3 e 4) LPS geradas (CLEMENTS, 2002)**

## 3.2 O que não é LPS?

Não pode ser considerada LPS:

- **Desenvolvimento de um sistema único com reuso:** Desenvolver um novo sistema reutilizando partes de outro sistema similar anterior não é LPS, pois LPS requer a produção de vários produtos de uma mesma família de uma forma deliberada.
- **Desenvolvimento baseado em componentes:** LPS requer um desenvolvimento baseado em componentes, mas é necessário que os componentes estejam de acordo com a arquitetura da linha de produção. Apenas desenvolver um componente reutilizável não significa que é um LPS.
- **Reuso de software em API's:** Construir um software reutilizando certa API de domínio ou aplicação não é LPS, É necessário que essa API tenha sido definida para uma arquitetura de visando uma linha de Produção.
- **Releases e versões de um mesmo produto:** LPS produz múltiplos produtos similares ao mesmo tempo. Porém cada um tem seus releases e versões.

## 3.3 Estratégias de desenvolvimento de uma LPS

Existem 3 estratégias diferentes de adoção da abordagem de LPS que podem ser usadas em seu desenvolvimento: pró-ativa, extrativa e reativa.

- **Pró-ativa:** nessa estratégia está voltada para o desenvolvimento de todos os artefatos da LPS, partindo da análise, projeto e implementação do domínio e indo posteriormente para a engenharia da aplicação. Esta estratégia considera que a LPS atenda a maior quantidade de produtos existentes de certo domínio.
- **Extrativa:** Nesta estratégia a LPS é desenvolvida a partir de um conjunto de softwares existentes, isto é, extraindo as características comuns e variáveis desse software.
- **Reativa:** a LPS e desenvolvida de forma incremental. Assim a LPS vai sendo ampliada quando existem demandas de novos requisitos ou produtos. Os Ativos são desenvolvidos à medida que forem necessários.

## 3.4 Atividades essenciais para uma linha de produtos de software

O *Software Engineering Institute* (SEI, 2007) estabeleceu as atividades essenciais para uma Linha de Produto de Software. Essas atividades são o

Desenvolvimento do núcleo de artefatos, ou Engenharia de domínio, o Desenvolvimento do produto, ou Engenharia da aplicação, e o Gerenciamento da linha de produto. A Figura 3-2 ilustra a ligação entre essas atividades. Cada círculo representa uma das atividades essenciais. Todos os três são ligados entre si e em movimento contínuo, mostrando que todos eles são essenciais, indissociáveis, e altamente interativo.

Cada círculo representa uma das atividades essenciais. Todos os três são ligados entre si e em movimento contínuo, mostrando que todos eles são essenciais, indissociáveis, e altamente interativo.


**Figura 3-2: Atividades essenciais para uma linha de produto de software**
**Fonte: (SEI, 2007)**

As setas rotativas indicam não só que os ativos são usados para desenvolver produtos, mas também que as revisões dos principais ativos existentes ou mesmo ativos novos podem, e na maioria das vezes não, evoluir do desenvolvimento do produto. Em alguns contextos, os produtos já existentes são extraídos de ativos genéricos, talvez, uma especificação de requisitos, uma arquitetura, ou componentes de software, que são guardados no repositório de ativos do núcleo da linha de produtos.

O repositório de ativos do núcleo tem um papel fundamental, pois nele ficam guardados os artefatos que serão reaproveitados para o desenvolvimento de cada produto da família de *softwares.*

### 3.4.1 Desenvolvimento do núcleo de artefatos

O objetivo da atividade de desenvolvimento do núcleo de artefatos é estabelecer uma infraestrutura central que será reutilizada pelos produtos gerados a partir da LP. Os artefatos de entrada para esta atividade são fornecidos pela engenharia de domínio, por exemplo:

• **As restrições do produto:** O que existem de semelhanças e variações entre os produtos que constituem a linha de produtos? Quais são as características comportamentais que eles oferecem? Quais são as características que as previsões de mercado e tecnologia dizem que vai ser benéfico no futuro? Quais os padrões comerciais? Quais os limites de desempenho que eles devem atender? Com que sistemas externos devem eles interagir? Que requisitos de qualidade (tais como disponibilidade e segurança) são impostos? Todas essas perguntas precisam ser respondidas para que se possa ter uma LPS qualificada, visto que, quanto mais respostas apresentadas maior será o entendimento sobre o produto no domínio.

• **As restrições de produção:** Deve ser um novo produto apresentado ao mercado em um ano, um mês ou um dia? Que capacidade de produção deve ser dada aos engenheiros em campo? Quais as normas específicas da empresa para os processos de desenvolvimento de software devem ser seguidas durante a produção de produto? Essas perguntas e outras irão orientar as decisões sobre, por exemplo, se é melhor investir em um ambiente gerador ou depender de codificação manual. Essas respostas, por sua vez, irão orientar as decisões sobre que tipo de mecanismos de variação para fornecer em ativos fundamentais e que processo de produção será utilizado para produtos e, eventualmente, codificado no plano de produção.

• **A estratégia de produção:** A estratégia de produção é a abordagem global para a realização de ambos os bens essenciais e produtos. Irá à linha de produto ser construído proativamente (começando com um conjunto de ativos do núcleo), reativamente (começando com um conjunto de produtos e generalizar seus componentes para produzir o núcleo de ativos da linha), ou utilizando uma combinação dos dois (ver "todos os três juntos")? A estratégia de produção determina a gênese da arquitetura e seus componentes associados e o caminho para seu crescimento. Informado pelas restrições do produto e as restrições de

produção, a estratégia de produção também impulsiona o processo pelo qual os produtos serão desenvolvidos a partir dos principais ativos.

• **Ativos preexistentes:** os artefatos existentes como, por exemplo, componentes, especificações ou partes legadas do domínio arquivadas para a sua futura reutilização. Sistemas legados e produtos existentes incorporam conhecimento de uma organização de domínio e/ou definir a sua presença no mercado. A arquitetura de linha de produto, ou pelo menos parte dela, pode pedir emprestado intensamente arquiteturas de projetos comprovados de sistemas legados ou produtos relacionados existentes. Os componentes podem ser extraídos de sistemas legados. Esses componentes podem representar a propriedade intelectual em domínios relevantes e, portanto, tornam-se os principais candidatos para os componentes do núcleo, base de ativos. O software e ativos da organização estão disponíveis desde o início do esforço de linha de produtos? Existem bibliotecas, ***frameworks***, algoritmos, ferramentas, componentes e serviços que podem ser usados? Existem processos de gestão técnica, modelos de financiamento e recursos de treinamento que podem ser adaptadas facilmente para a linha de produtos? Através de uma análise cuidadosa, uma organização determina o que é mais apropriado para usar. No entanto, os bens preexistentes não estão limitados aos bens que foram construídos pela organização da linha de produto. Externamente softwares disponíveis, tais como, serviços Web e produtos de código aberto, bem como, as normas, padrões e frameworks, são exemplos de ativos pré-existentes que podem ser importados de fora da organização e usados para uma boa vantagem.

Esta atividade de desenvolvimento do núcleo de artefatos tem como artefatos de saída:

• **O Escopo da linha de produto:** é uma descrição dos produtos que constituem a linha de produtos ou que a linha de produto é capaz de incluir. Em sua forma mais simples. Geralmente esta descrição é colocada em termos de semelhanças e diferenças dos produtos. Essas semelhanças e diferenças podem incluir as características ou operações que oferecem o desempenho ou outros atributos de qualidade que apresentam as plataformas em que eles funcionam.

- **Núcleo ou ativo base:** Artefato ou recurso que pode ser usado em um ou mais produtos de uma LPS. Pode ser uma arquitetura, um componente de

software, um modelo de processo, ou qualquer outro resultado que possa ser utilizado para a construção de um sistema e reaproveitado.

Como já é sabido, os ativos essenciais carregam em si uma arquitetura que os produtos da linha de produtos compartilharão, como também alguns componentes de software que são desenvolvidos para reutilização sistemática do outro lado da linha de produto.

Componentes de software também podem carregar consigo os planos de teste, casos de teste e todo o tipo de documentação do projeto. Requisitos, especificações e modelos de domínio são bens essenciais, como é a declaração do escopo da linha de produto. Serviços da Web também constituem ativos essenciais. Além disso, qualquer infraestrutura de produção, tais como linguagens específicas de domínio, ferramentas, geradores, e os ambientes são ativos fundamentais também.

Cada núcleo ativo deve pode ter um processo associado que especifica como ele irá ser utilizado no desenvolvimento de produtos reais. Por exemplo, o processo com as exigências da linha de produto, daria ao ativo a identificação dos requisitos para um produto individual. Este processo poderia simplesmente dizer, quais os requisitos da linha de produtos são os requisitos básicos, verificar a permissão de uma variação, validar que variações e extensões são suportadas pela arquitetura.

O processo também pode informar uma ferramenta de apoio para realizar essas etapas. Restrições de produtos, restrições de produção, bem como a estratégia de produção, também são exemplos de processos que podem estar associados com ativos do repositório.

### 3.4.2 Desenvolvimento do produto

A atividade de desenvolvimento de produto depende das duas entradas: o escopo linha de produtos e os bens essenciais.

Na Figura 3-2 as setas rotativas indicam iterações e as relações entre as atividades essenciais. Por exemplo, a existência e disponibilidade de um produto particular pode também afetar os requisitos para um produto subseqüente.

Como outro exemplo, a construção de um produto que tem em comum uma característica anteriormente não reconhecida por outro produto já na linha de

produtos, vai criar uma pressão para atualizar os ativos essenciais e fornecer uma base para a exploração de que para os produtos comuns no futuro.

As entradas para a atividade de desenvolvimento de produtos são as seguintes:

- A descrição para o produto particular, muitas vezes expresso como uma variação de alguma descrição do produto genérico contido no âmbito linha de produtos (por exemplo, uma descrição genérica é, ela própria, um ativo núcleo).
- O escopo linha de produtos, que indica se é viável incluir o produto na linha de produtos.
- Os ativos do núcleo a partir do qual o produto é construído.
- O plano de produção, que detalha como os bens essenciais estão a ser utilizados para construir o produto

Vale lembrar que os construtores de produtos também têm a obrigação de dar *feedback* sobre quaisquer problemas ou deficiências encontradas com os bens essenciais, de modo que o núcleo base de ativos permanece saudável e viável.

### 3.4.3 Gerenciamento da linha de produto

A atividade de gerenciamento de LP deve garantir que todas as atividades técnicas sejam realizadas de acordo com um planejamento coordenado.

- Gerenciamento técnico: coordena as atividades de desenvolvimento do núcleo de artefatos e desenvolvimento do produto para garantir que as equipes de desenvolvimento sigam os processos definidos para a LP e coletem dados suficientes para acompanhar o progresso desta.
- Gerenciamento organizacional: garante que as unidades organizacionais recebam os recursos corretos (ex. treinamento) em quantidades suficientes. Uma das ações mais importantes da atividade de gerenciamento da LP é a criação de um plano de adoção que descreva o estado desejado da organização e uma estratégia para alcançar tal estado.
- Gerenciamento de LP também deve estabelecer como as atividades de desenvolvimento do núcleo de artefatos e desenvolvimentos dos produtos devem interagir para permitir a evolução da LP e o gerenciamento das semelhanças e das variabilidades de cada artefato da LP.

## 3.5 Vantagens e desvantagens do modelo de linha de produto de software

Assim como a maioria das inovações tecnológicas possuem seus pontos negativos e positivos. A LPS não é muito diferente, apesar das vantagens serem mais abundantes do que suas desvantagens. Ao mesmo tempo em que se observa um aumento de lucratividade e qualidade, é notório a dificuldade se implementar uma LPS, visto que o próprio gerente da LPS pode criar padrões desapropriados e não conduzir bem a mesma.

### 3.5.1 Vantagens da LPS

Quando se fala em LPS, é possível visualizar uma série de benefícios encontrados em diversas empresas que adotam tal modelo:

- **Lucratividade:** O repositório de ativos permite que a organização desenvolva produtos voltados para um segmento específico de mercado. O benefício dessa focalização é observado no aumento da participação de mercado e aumento da lucratividade. Além de economia na produção de novos produtos, já que todos os produtos compartilham *features* em comum, estas não necessitarão ser desenvolvidas a cada novo produto, diminuindo o tempo de desenvolvimentos e custos de produção.
- **Qualidade:** Uma redução no número de defeitos relatados é comum em sistemas desenvolvidos em LPS. A qualidade também pode ser medida em termos de redução do tempo de correções e da redução do efeito *ripple* [6]. Sabe-se que os testes em fase de desenvolvimento do produto não são suficientes para descobrir todos os problemas que poderão ocorrer durante a vida do software. Geralmente quando o produto é colocado no mercado de trabalho sempre aparecerão ***bugs***[7] quando um usuário final usa o sistema, como o modelo LPS possui variações de softwares que compartilham *features* em comum, consegui-se assim um número maior de usuários utilizando o sistema, logo a probabilidade de aparecer um erro será maior, e sendo esse erro corrigido, o sistema estará ficando cada vez com mais qualidade em um tempo mais curto.

---

[6] *Ripple: geração de novos defeitos a partir de correções executadas*
[7] *Bugs: problemas que ocorrem quando usamos um software*

- **Desempenho de produtos de software:** A utilização de ativos reutilizáveis aumenta a desempenho em relação ao desenvolvimento tradicional, especialmente com o aumento da maturidade da linha, o que faz com que os ativos estejam cada vez mais otimizados.
- **Produtividade:** A equipe de desenvolvimento pode ser reduzida (diminuindo os gastos), o custo total de desenvolvimento é cortado consideravelmente, o cronograma é reduzido (maior velocidade de lançamento), o sistema possui uma flexibilidade documentada, o que facilita o atendimento das solicitações de modificações do cliente.
- **Satisfação profissional:** Os desenvolvedores relatam que o trabalho braçal já foi realizado (desenvolvimento dos ativos de software), assim eles podem se concentrar em atividades mais interessantes, como o aperfeiçoamento e/ou inovação de elementos específicos.

### 3.5.2 Desvantagens da LPS

Um projeto que envolva uma implantação de uma linha de produto de software pode ser ponderado como um projeto de inovação tecnológica, ou seja, um projeto de adaptação a uma nova maneira de fazer negócio. Como toda mudança tecnológica, esse tipo de projeto deve ter uma avaliação da situação atual da empresa, uma articulação do estado desejado e a elaboração de um plano para atingir este estado. Especificamente no caso de linha de produto, por ser um modelo que interfere diretamente na maneira de trabalhar da empresa, fatores não tecnológicos devem ser considerados, como a adaptabilidade das pessoas, o tipo de treinamento necessário e a preparação do cliente para a nova maneira de trabalhar.

As dificuldades de implantação de uma linha são diversas e podem vir de várias fontes. Conforme mencionado, o processo envolve tanto pessoas como tecnologias e pode sofrer muita resistência dentro da organização. Existem alguns problemas que foram encontrados por (COHEN, 2002), são eles:

- **Líder sem compromisso:** para o processo de adaptação sugere-se a escolha de um líder, uma pessoa que acredita nos princípios do modelo e que estará monitorando o processo e deixando as pessoas envolvidas motivadas. O líder estará comunicando e dando apoio aos desenvolvedores em momentos de dificuldade evitando que eles se esqueçam do modelo. Caso esta pessoa não receba a

autoridade exigida, não confie no modelo, é muito difícil continuar com o foco e a confiança durante o processo, existindo assim um grande risco de desistência.

- **Padronizações desapropriadas:** Existe uma tendência nas organizações de assumir que a adoção de padrões irá guiar automaticamente para a implantação de uma linha de produto. A institucionalização prematura e impensada de padrões, muitas vezes obsoletos, pode limitar as opções de tecnologia da linha, prejudicando o escopo da mesma. Os padrões devem ser estudados caso a caso e, apesar de serem importantes para a linha devem ser discutidos antes de suas implementações.
- **Evolução da abordagem:** Esse fator também poderia ser chamado de melhoria contínua, ou seja, caso o processo da linha de produto não seja revisado e atualizado periodicamente, suas práticas irão tornar-se obsoletas e inapropriadas às novas necessidades que surgirão. Neste caso, novas práticas não previstas irão surgir informalmente devido às necessidades e poderão comprometer todo o modelo.

# 4 APLICAÇÃO DA LPS NO DESENVOLVIMENTO DE UMA CALCULADORA

No estudo de caso proposto neste trabalho, será relaizado o desenvolvimento de uma arquitetura para calculadoras, que existem vários tipos, mas estará em foco somente dois: Calculadora Simples (Soma, Subtração, Multiplicação e Divisão), Calculadora Avançada (Raiz quadrada, cálculo exponencial e etc). É importante destacar os passos a serem seguidos no processo de desenvolvimento de uma LPS para calculadoras.

## 4.1 Engenharia de requisitos de domínio

Não foi destacada a fase da gestão de produtos visto que já se tem um conhecimento do ramo do mercado que se deseja alcançar com a LPS, no caso, calculadoras. Portanto a fase inicial será a engenharia de requisitos de domínio. Confira na Tabela 4-1 os requisitos para o estudo de caso em questão. Será utilizada a forma de desenvolvimento pró-ativa.

**Tabela 4-1: Parte dos requisitos da LPS calculadora**

| CÓDIGO | DESCRIÇÃO |
|---|---|
| C01 | O sistema deverá permitir que o usuário faça operações matemáticas |
| C02 | Existe um número enorme de operações matemáticas, onde dependendo da área do profissional, ele pode usar algumas operações e outras não |
| C03 | É observado que existem várias operações (Somar, Subtrair, multiplicar, Dividir, Raiz quadrada, exponencial e entre outras que podem surgir) |
| C04 | O operador do sistema será na maioria das vezes, qualquer profissional da área matemática dos diversos ramos |
| C05 | Todos os usuários usam operações básicas (Somar, Subtrair, Multiplicar e Dividir) |

## 4.2 Desenho do domínio

Com os requisitos já em mãos será feito a demonstração de variabilidades utilizando o caso de uso visto no ANEXO A. Acompanhe na Figura 4-1 o resultado do diagrama de caso de uso com variabilidade.

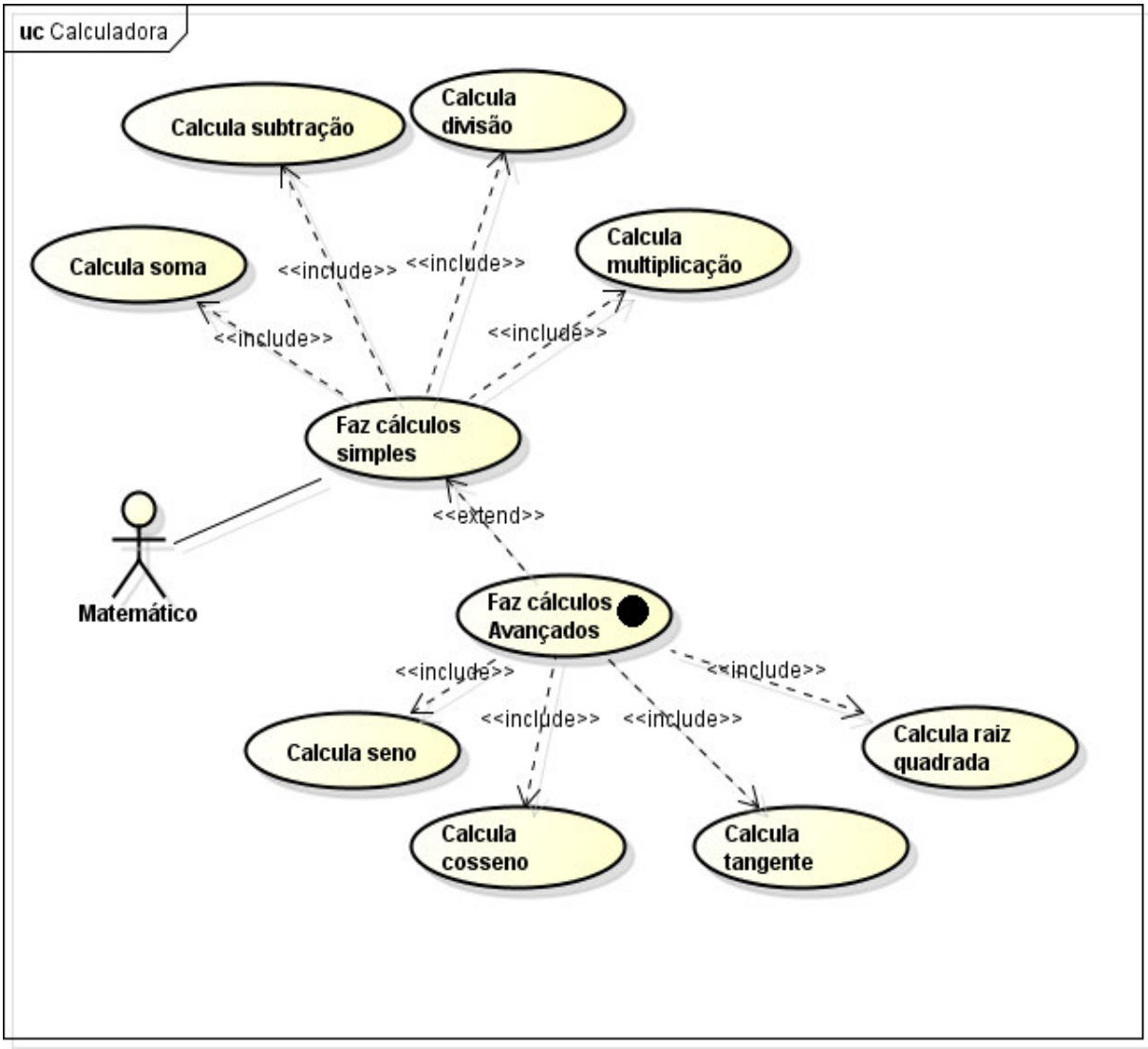


**Figura 4-1: Diagrama de caso de uso com variabilidade para a LPS**

Verifique no caso de uso explanado que de acordo com os requisitos, os usuários sempre usam os cálculos simples (Soma, subtração, multiplicação e divisão), sendo opcional o uso de cálculos avançados (raiz quadrada, exponencial, seno, cosseno, tangente e etc).

### 4.2.1 Descrição do cenário

Será utilizada a representação de cenários por etiqueta apresentada no ANEXO B, onde fica possível demonstrar o funcionamento de variabilidade dentro de um cenário de caso de uso, acompanhe o resultado após a aplicação da técnica.

**Tabela 4-2: Exemplo de cenário de um caso de uso com representação por etiquetas**

**Id:** CA01
**Ator:** Matemático
**Objetivo:** realizar { V0 } utilizando a calculadora
**Pré-condição:** o usuário ter iniciado o programa da calculadora
**Cenário de sucesso:**
- Quando a calculadora é iniciada o sistema mostra as possíveis funções de { V0 }
- Para utilizar as funções de { V0 } o usuário deve proceder:
- {V2}
- O usuário digita os valores e escolhe qual função deseja utilizar e o sistema fornece o resultado

---

V0: alternative
V0: 1. Cáculos simples
2. Cálculos avançados

---

V2: parâmetro
if V0=1 then
procedure_A:
- digite o primeiro valor no campo1
- digite o segundo valor no campo2
- escolha a função e confira o resultado no campo3

else if V0=2 then
procedure_A:
- pressione o botão ADVANCED
- digite o valor no campo1
- escolha a função e confira o resultado no campo3

### *4.2.2 Configuration knowledge*

Também chamado de arquivo de configuração, como já falado no início deste trabalho, é extremamente necessário, pois este arquivo determinará quais os produtos estão fora do escopo de criação da linha de produto. Usando o exemplo da calculadora em questão, pode-se deduzir o seguinte arquivo de configuração, conforme a Tabela 4-3.

**Tabela 4-3: Arquivo de configuração da Calculadora**

| Feature | Tarefa |
|---|---|
| Calculadora | Selecionar o cenário CA01 |
| Funções Simples | Não utilizar os componentes ModoAvancado, Seno, Cosseno, Tangente, RaizQuadrada<br>E utilizar Soma, Subtracao, Divisao e Multiplicacao |
| Funções Avançadas | Utilizar todos os componentes válidos na feature Funções Simples acrescido dos componentes ModoAvancado, Seno, Cosseno, Tangente, RaizQuadrada |

Observe que o arquivo de configuração tem que corresponder ao caso de uso gerado, de acordo com o arquivo de configuração criado, fica claro observar os produtos que fazem parte do escopo dessa linha de produto, por exemplo, uma calculadora que tenha "**apenas"** funções avançadas (Seno, Cosseno, Tangente, RaizQuadrada) não faz parte do escopo da linha de produtos, ou seja, será impossível criar um produto que atenda essa característica, pois toda calculadora avançada deve também realizar funções simples (Somar, Subtrair, Dividir e Multiplicar).

#### 4.2.2.1 Modelo de features

Com o diagrama de caso de uso expressando as variabilidades de uma LPS, será viável a elaboração do modelo de features, para poder verificar quais produtos são válidos na LPS.

Existem vários *frameworks* que podem demonstrar de uma forma gráfica os produtos que são válidos e os não válidos em uma linha de produto de software, essa é a importância da criação do modelo de *features**.*** No caso em questão será utilizado o FMP (***Feature Modeling Plug-in***) para implementar o modelo de *features*. Este *framework* está detalhado no ANEXO C.

Nessa ferramenta todas as restrições do arquivo de configuração poderão ser implementadas e demonstradas de uma forma gráfica, ficando mais difícil cometer erros ao gerar novos produtos para a LPS e mais fácil visualizar quais os possíveis produtos válidos para aquela LPS.

Acompanhe na Figura 4-2 que as restrições não permitem a criação de uma calculadora "**apenas"** com funções avançadas, pois todas as vezes que uma nova calculadora da LPS utilizar uma função avançada obrigatoriamente pela restrição, deverá utilizar todas as funções simples e o componente "Modo Avançado", conforme mostrada na Figura 4-2 foi criado o componente "Modo Avançado" para que o layout fique mais organizado agrupando as funções avançadas.

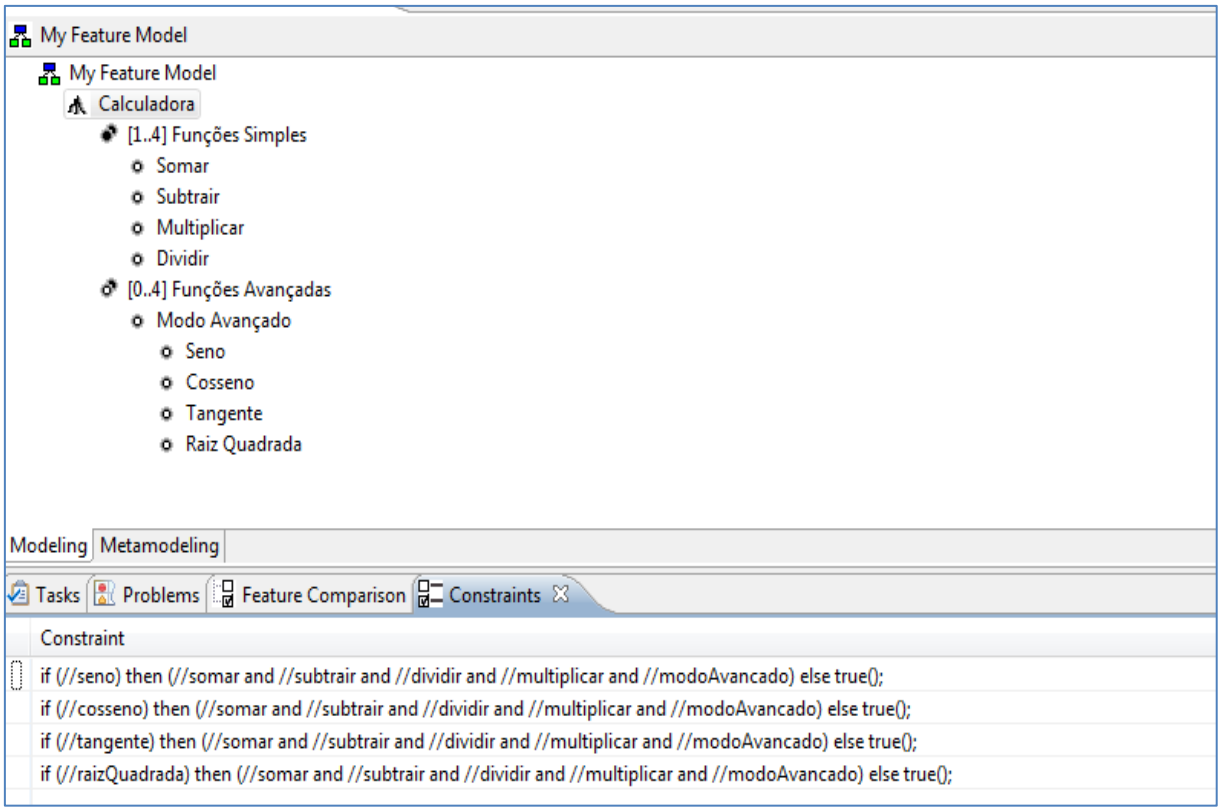


**Figura 4-2: Modelo de *features* com as restrições de produtos válidos**

Na Figura 4-3 ficam claras duas configurações válidas para esta LPS, ou seja, duas calculadoras que poderiam ser produzidas nessa LPS, porém não são as únicas, poderia existir também, uma calculadora que apenas realiza soma entre outras que não são impedidas pelas restrições.

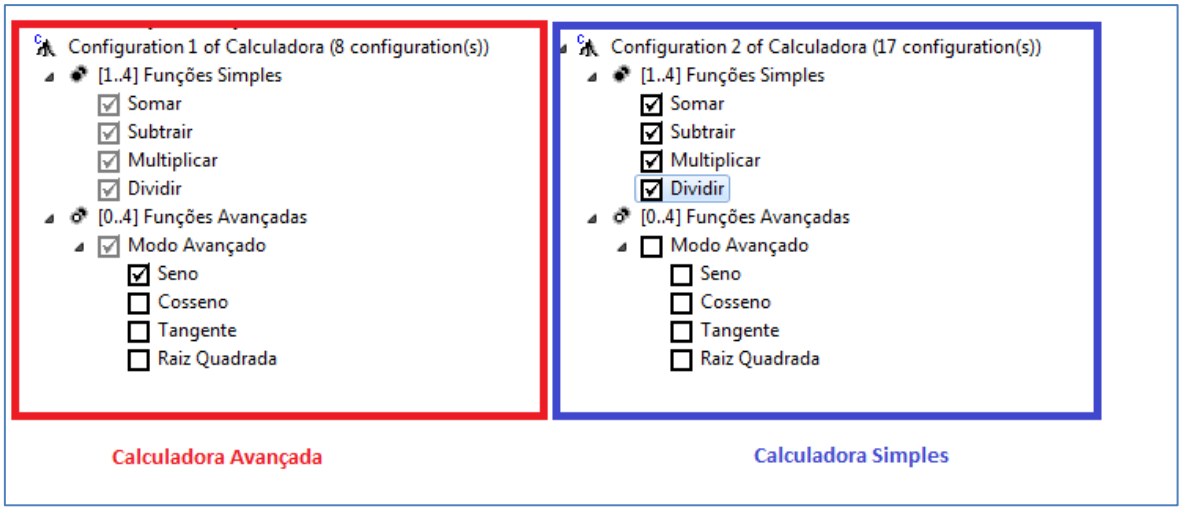


**Figura 4-3: Produtos válidos para a LPS calculadora**

## 4.3 Realização de domínio

Como se pode observar até chegar nessa fase, já se pode ter uma ideia das variabilidades que a LPS pode atingir, quais produtos são viáveis com seu escopo criado, porém ainda falta implementar os componentes imaginados, quando se fala em implementar fala-se em desenvolvimento, criar um código fonte que atenda as necessidades expressadas pela LPS.

Nessa etapa é de suma importância utilizar algum framework que posso oferecer tais facilidades, por exemplo, para o programador já está mais que evidente que ele precisará utilizar uma forma de programação diferente, na qual ele poderá criar métodos, botões e classes em tempo de execução, pois está se tratando de um software (arquitetura) que pode se comporta de várias formas, ora mostrando determinados botões ora não mostrando botões, por exemplo.

Sendo assim foi utilizada a “Programação orientada a aspecto”, como explicado no ANEXO D. Acompanhe na

Figura 4-4 como ficou o desenvolvimento da arquitetura reutilizável na IDE Eclipse.

Na cor AZUL se encontra a classe “Nucleo” que é a arquitetura que será reutilizada em todas as calculadoras que fazem parte dessa linha de produto, dependendo do arquivo de configuração. Em VERMELHO estão os componentes codificados, observe que são as *features* presentes no Modelo de *Features* da

Figura 4-4. Em VERDE está um exemplo de implementação de um componente (Aspecto), no caso foi mostrado o componente OperacaoAdicao.

A classe “**Nucleo**” possui um método chamado de “`addComponents ()`”, que ao ser chamado irá disparar dentro de cada aspecto implementado um método que irá adicionar botões no layout, em tempo de execução. Sendo assim, se a classe aspecto estiver dentro do projeto, irá colocar no layout seu respectivo botão, por exemplo: se o aspecto “OperacaoAdicao” estiver dentro do projeto irá criar na tela o botão adição.

Como se pode observar, a existência de cada ação na calculadora só depende da existência do respectivo aspecto dentro do projeto, esta foi a forma de programação escolhida neste trabalho. No caso de todos os aspectos estarem dentro do projeto, tem-se desenvolvida uma calculadora avançada, observe a Figura 4-5.

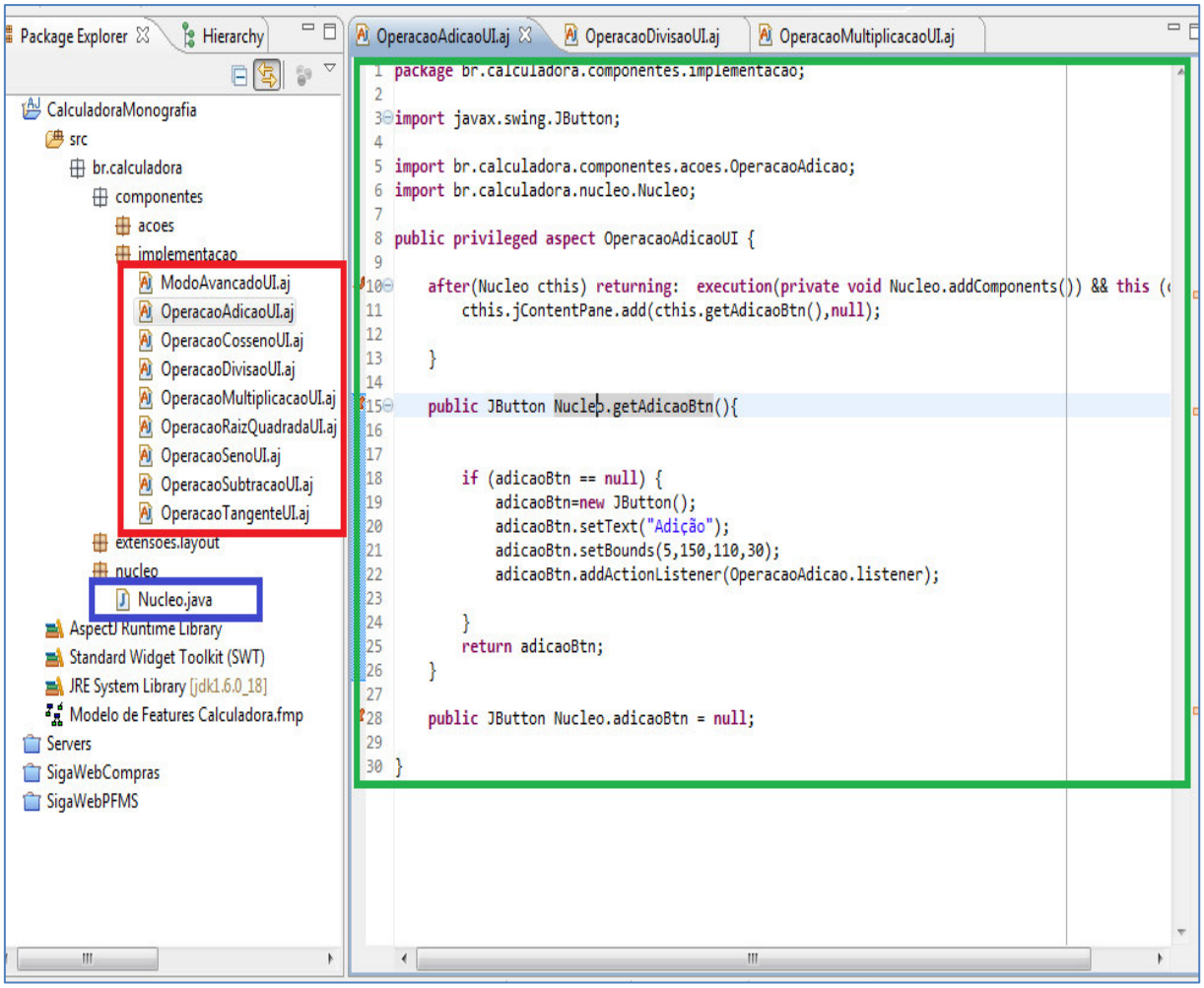


**Figura 4-4: LPS implementada na IDE Eclipse**

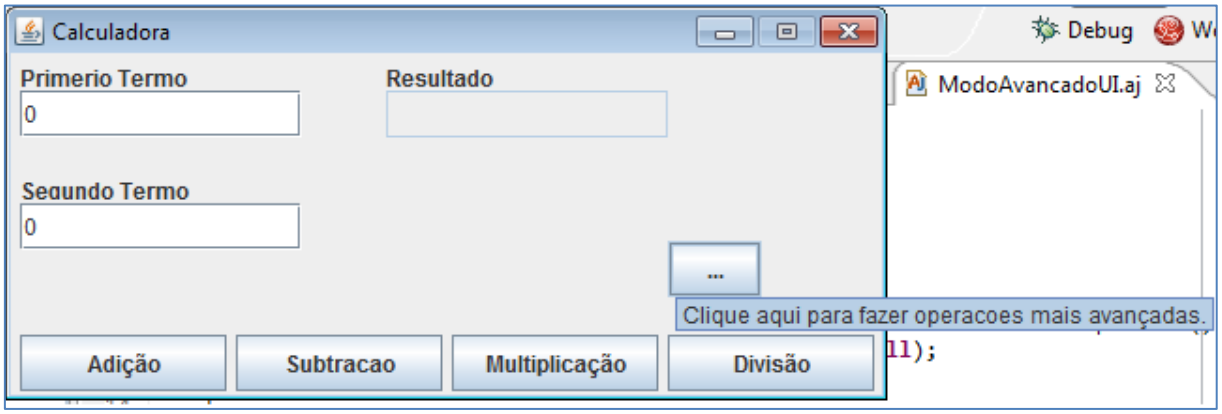


**Figura 4-5: Exemplo calculadora avançada**

Caso seja desejada uma calculadora que apenas realize soma e subtração deverá estar presente no projeto somente os componentes

OperacaoAdicao e OperacaoSubtracao, então serão excluídos os demais aspectos do projeto, restando apenas OperacaAdicao e OperacaoSubtracao, como foi feito Figura 4-6.

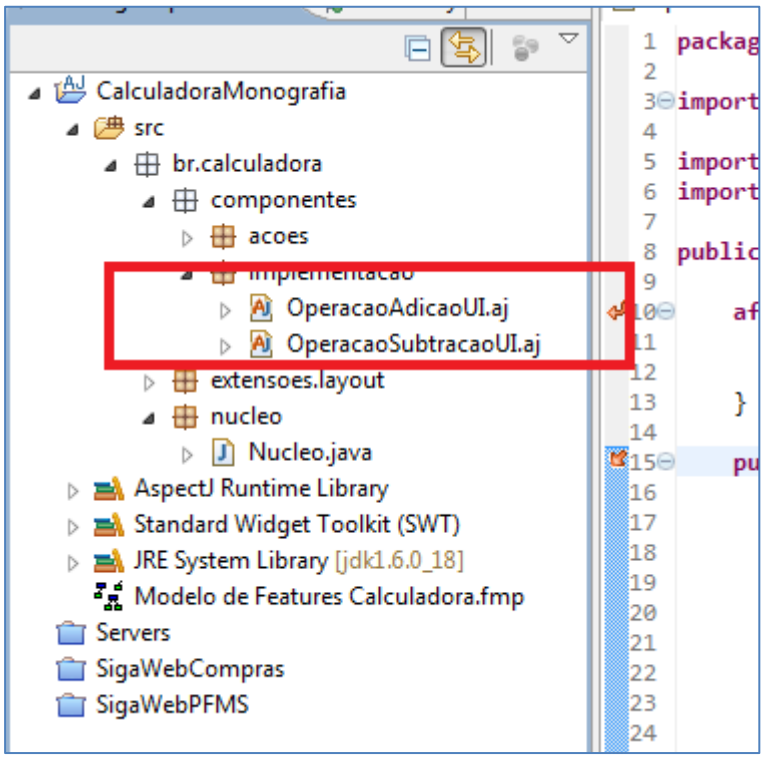


**Figura 4-6: Exclusão de todos os componentes diferentes da Adição e Subtração**

Ao executar o projeto novamente será mostrada uma calculadora simples, como mostrada na Figura 4-7.

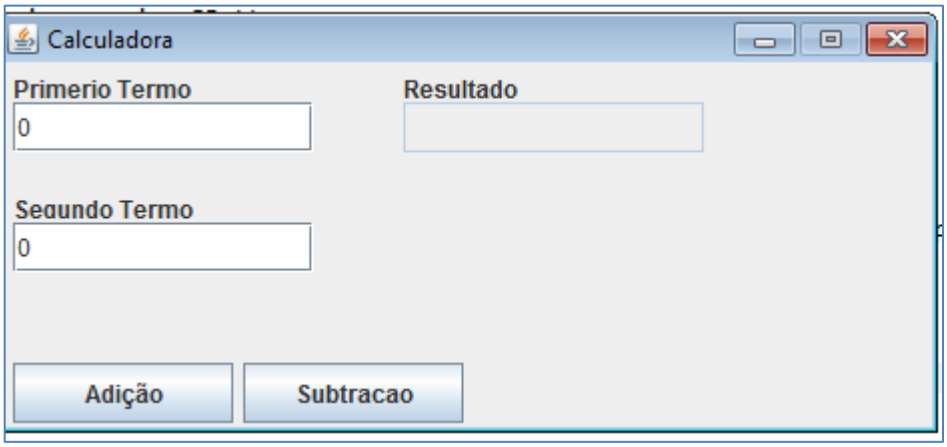


**Figura 4-7: Exemplo Calculadora Simples**

Seguindo a mesma lógica poderão ser criados diferentes tipos de calculadora, ou melhor, uma LPS de calculadoras, lembrando que as regras de exclusão de aspectos válidos e combinações válidas dependem do *Configuration knowledge* apresentada anteriormente, para que sejam criadas todas as calculadoras válidas na LPS.

# 5 CONCLUSÃO

Este trabalho apresentou dois conceitos no ramo da tecnologia: Engenharia de Domínio e Linha de Produto de Software. Como foi falado, a Engenharia de Domínio possui alguns processos em seu desenvolvimento que lembram a Engenharia de *Software,* porém a Engenharia de Software visa atender uma aplicação dentro de uma classe de aplicações, já a Engenharia de Domínio visa desenvolver uma arquitetura que possa ser reutilizada, atendendo assim, o maior número possível de aplicações dentro de um domínio, ou em outras palavras, visa desenvolver uma arquitetura que será compartilhada por uma Linha de Produto de *Software*.

Por fim, o domínio de Calculadoras foi escolhido, a fim de servir como estudo de caso da proposta apresentada durante este trabalho. Primeiramente, foram identificados os requisitos específicos dessa classe de aplicações, para que um processo de desenvolvimento adequado possa ser definido, o que é feito a partir da instanciação do modelo genérico de desenvolvimento de uma arquitetura para e com reuso proposto, e finalizado com a codificação dessa arquitetura.

Foram seguidos alguns passos que já são conhecidos na engenharia de *software*, como o modelo de casos de uso, cenários de caso de uso e requisitos, porém sempre visando a generalização, para atender o maior número possível de sistemas nesse domínio. A partir do reuso dessa infraestrutura, bem como de outros componentes passíveis de reutilização, construiu-se então dois exemplos dentre as várias aplicações possíveis nesse domínio: Calculadora Simples e Calculadora Avançada.

Foi observado que a LPS facilita e agiliza o desenvolvimento de novos produtos que serão colocados no mercado, já que todos os produtos compartilham uma mesma arquitetura unida com algumas particularidades (*features*).

Observou-se que apesar da sua grande agilidade, a LPS sofre algumas barreiras: como a desconfiança em relação ao novo modelo de desenvolvimento e a falta de disposição de revisar e atualizar periodicamente a LPS.

Verificou-se que o desenvolvimento de uma calculadora utilizando LPS, possui um esforço inicial de implementação maior do que o esforço usado para desenvolver essa mesma calculadora sem o uso de LPS.

Para trabalhos futuros sugere-se: um estudo sobre a Engenharia da Aplicação, que irá detalhar e organizar o desenvolvimento de cada produto dentro de uma LPS; Desenvolver uma aplicação que poderia gerar um modelo de *features* associado com o projeto do eclipse, onde uma exclusão de *features,* deletaria as classes correspondentes no projeto.

## BIBLIOGRAFIA

BACELO, D. A. (07 de 2009). *PUCRS.* Acesso em 19 de 07 de 2012, disponível em Universidade Católica do Rio Grande do Sul: http://www.pucrs.br/edipucrs/XSalaoIC/Ciencias_Exatas_e_da_Terra/Ciencia_da_Computacao/70935-CASSIA_ZOTTIS_ALMEIDA.pdf

CLEMENTS, P. (2002). In: Addison-Wesley, *Software Product Lines: Practices and Patterns* (p. 563). Boston.

COHEN, S. (2002). *Product Line State of Report.*

DURSCKI, R. C. (2004). Linhas de Produto de Software: riscos e vantagens de sua implantação. *VI Simpósio Internacional de Melhoria de Software*, (p. 159). São Paulo. *FEATURE MODELING PLUG-IN.* (s.d.). Acesso em 01 de 08 de 2012, disponível em fmp: Feature Modeling Plug-in: http://gp.uwaterloo.ca/book/export/html/16

FROHLICH, A. A. (2001). *UNIVERSIDADE FEDERAL DE SANTA CATARINA*. Acesso em 2012 de 09 de 19, disponível em PROGRAMAÇÃO ORIENTADA A ASPECTOS: http://www.lisha.ufsc.br/teaching/sce/ine5612-2001-2/work/aop.html

GUIZZARDI, G. (2000). Desenvolvimento para e com reuso: Um estudo de caso no domínio de vídeo sob demanda. VITÓRIA.

IAN, S. (2004). Software Engineering.

KRUEGER, C. W. (Julho de 2011). Acesso em 01 de 03 de 2012, disponível em http://www.softwareproductlines.com/introduction/concepts.html

NETO, C. D. (2006). *Utilização de Aspectos como Mecanismo de Implementação de Variabilidade para Instanciação de Componentes de Negócio Reutilizáveis.* Minas Gerais.

PACIOS, S. F. (2006). *Uma abordagem orientada a aspectos para o desenvolvimento de linhas de produto de software.* São Carlos.

*PROJETO YANA*. (2010). Acesso em 2012 de 09 de 20, disponível em PROJETO YANA: https://sites.google.com/site/projetoyanaufpb/revista/lps-analise-modelagem-notacoes-ferramentas-e-rastreabilidade/4-ferramentas-para-modelagem-de-features/4-2-feature-modeling-plug-in-fmp

*SEI*. (2007). Fonte: http://www.sei.cmu.edu/productlines/frame_report/PL.essential.act.htm

SEI. (2007). *Sofware Product line | Overview*. Acesso em 21 de 03 de 2012, disponível em Sofware Engineering Intitute: http://www.sei.cmu.edu/productlines/

SILVEIRA, M. C. (2006). *A Reutilização de Requisitos no Desenvolvimento e Adaptação de Produtos de Software.*

# ANEXOS

# ANEXO A - REPRESENTAÇÃO DE VARIABILIDADE EM CASO DE USO[8]

A variabilidade é usada para permitir variações nos requisitos funcionais e também nos requisitos não funcionais. É mais fácil entender a necessidade de variabilidade quando se observam as diferentes categorias de variabilidade, para ver como elas podem ser representadas e implementadas.

Observe algumas razões típicas para explorar a variabilidade em um componente de caso de uso:

- Variações de usuários ou interfaces de sistema – Diferentes tipos de usuários, ou diferentes especializações de casos de uso podem especificar interfaces de usuário diferentes. Também, instalações diferentes de um sistema ou ambientes de operação diferentes podem requerer interfaces de sistemas diferentes ou drivers customizados. Isto é observado, por exemplo, em caixas automáticos de bancos.

- Diferentes tipos de entidades referenciadas – Por exemplo, um caso de uso **Conta** pode ser uma conta de cheque ou conta conjunta. Tais variações frequentemente correspondem diretamente a um entidade no objeto modelo.

- Funcionalidades Alternativas e Opcionais – Variantes diferentes de casos de uso podem oferecer comportamentos alternativos ou opcionais.

- Variando restrições e regras de negócio – Variantes diferentes de caso de uso podem ter várias restrições na ordem em que as tarefas são processadas. Elas podem ter pré-condições diferentes ou restrições do que os casos podem executar e pode haver diferentes regras de negócio para serem aplicadas.

- Performance e diferenças de escalabilidade – Diferenças em requerimento de performance, restrições de tempo e um numero de atividades urgentes podem variar entre diferentes instalações de um sistema.

Dentre as principais soluções para representar variabilidade, estão às propostas por Jacobson, Griss e Jonsson (1997) para diagramas de casos de uso; Claub (2001b) para modelos de *features* e diagramas de classes e Gomaa e Webber (2004) para artefatos da UML. Jacobson, Griss e Jonsson (1997) propõem o uso de uma marca (“•”) para representar variabilidade em casos de uso. Porém, variabilidades em atores não são tratadas. A Figura A-1 apresenta um exemplo de representação de variabilidade em diagramas de casos de uso proposta por

[8] (NETO, 2006)

Jacobson, Griss e Jonsson em que o caso de uso “Agendar Aula” representa um ponto de variação e possui duas possíveis variantes, representadas pelos casos de uso “Agendar Aula Teórica” e “Agendar Aula Prática”. Em diagramas de casos de uso, a variabilidade é representada com o auxílio do estereótipo <<extend>>, além da marca “•”.

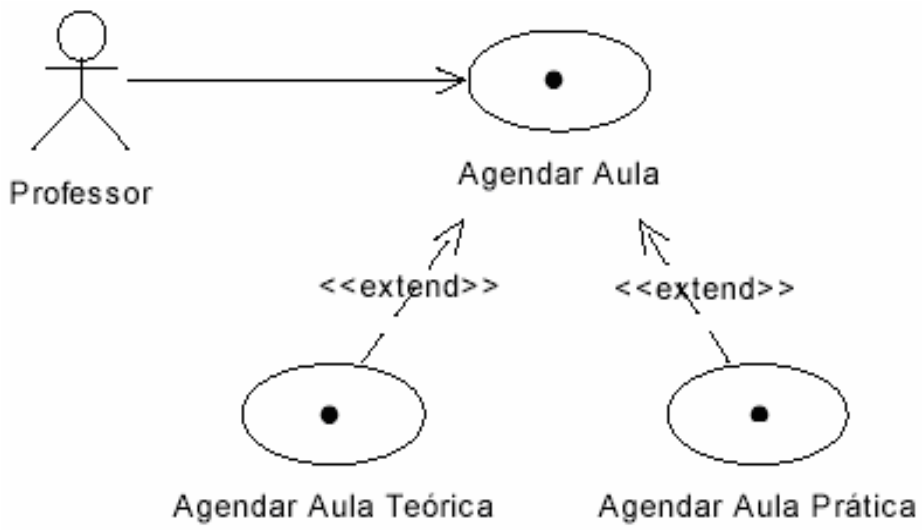


**Figura A-1: Exemplo de representação de variabilidade proposta por Jacobson, Griss e Jonsson**

# ANEXO B - REPRESENTAÇÃO DE VARIABILIDADE NO MODELO DE CASOS DE USO EM LINHAS DE PRODUTOS[9]

A representação da variabilidade de requisitos de linhas de produtos em modelos de casos de uso tem sido apontada por vários desenvolvedores da área.

```
Primary Actor: the {[V0] 33-serie} mobile phone
Goal: play a game on a {[V0] 33-serie} mobile phone
        and record score
Preconditions: the function GAME has been selected
        from the main MENU

Main Success Scenario:
- The system displays the list of the {[V1] available} games
- The user select a game
- The system displays the logo of the selected game
- The user selects the difficulty level by
  following the {[V2] appropriate} procedure and press YES
- The system starts the game and plays it until it goes over
- The user records the score achieved and
  {[V3] possibly} send the score to Club Nokia via WAP
- The system displays the list of the {[V1] available} games
- The user presses NO

V0: alternative

V0: 1. Nokia 3310 model

     2. Nokia 3330 model

V1: optional
      if V0=1 then game1 or game2
      else if V0=2 then game1 or game2 or game3

V2: parametric
      if V0=1 then procedure-A: - press Select
        - scroll to Options and press YES
        - scroll to Difficulty Level and press YES
        - select the desired difficulty level, press YES
      else if V0=2 then procedure-B: - press Select
        - scroll to Level and press YES
        - select the desired difficulty level, press YES

V3: parametric
      if V0=1 then function not available
      else if V0=2 then function available
```

**Figura B-1: Exemplo de representação de variabilidade proposta por Jacobson, Griss e Jonsson**

[9] (SILVEIRA, 2006)

A maioria dos desenvolvedores usa duas abordagens para representar a variabilidade em *templates* de caso de uso. representação por etiqueta (*tag representation*) e representação nivelada (*flat representation*). A representação por etiqueta pressupõe a organização dos casos de uso em dois níveis: família e produto. A variabilidade é expressa nos cenários através de três tipos de etiquetas: alternativa, paramétrica ou opcional. A Figura B-1 ilustra um *template* de caso de uso com utilização de etiquetas.

Na representação nivelada (*flat representation*), a variabilidade é incluída diretamente no *template* do caso de uso como mostra a Figura B-2:

```
Primary Actor: The 3310/3330 mobile phone
Goal: play a game on a 3310/3330 mobile phone and record score
Preconditions: the funcextend_usecase.epstion GAME has been
                    selected from the main MENU

Main Success Scenario:
- 3310 model: The system displays game1 or game2
  3330 model: The system displays game1 or game2 or game3
- The user select a game
- The system displays the logo of the selected game
- The user selects the difficulty level by following this procedure:
   3310 model: press Select scroll to Level and press YES
                     select the desired difficulty level and press YES
   3330 model: press Select scroll to Options and press YES
                     scroll to Difficulty Level and press YES
                     select the desired difficulty level and press YES
- The system starts the game and plays it until it goes over
- The user records the score achieved
- 3330 model: send the score to Club Nokia via WAP
- 3310 model: The system displays game1 or game2
  3330 model: The system displays game1 or game2 or game3
- The user presses NO
```

**Figura B-2: Exemplo de um caso de uso com representação nivelada**

## ANEXO C – *FEATURE MODELING PLUGIN*[10]

É um plug-in do Eclipse que serve para edição e configuração de modelos de *features*. Esse plug-in pode ser usado no Eclipse autônomo ou em conjunto com o fmp2rsm que é uma implementação do protótipo do modelo de *features* com base em *templates* para o Modelo de Software Racional (*Rational Software Modeler* – RSM) ou Arquitetura de Software Racional (*Rational Software Architect* – RSA).

Fmp2rsm integra o FMP com o RSA e permite a modelagem da linha de produtos em UML e derivação automática de produtos. Sua última versão é a 0.7.0 e é um plug-in com fácil instalação e notação própria.

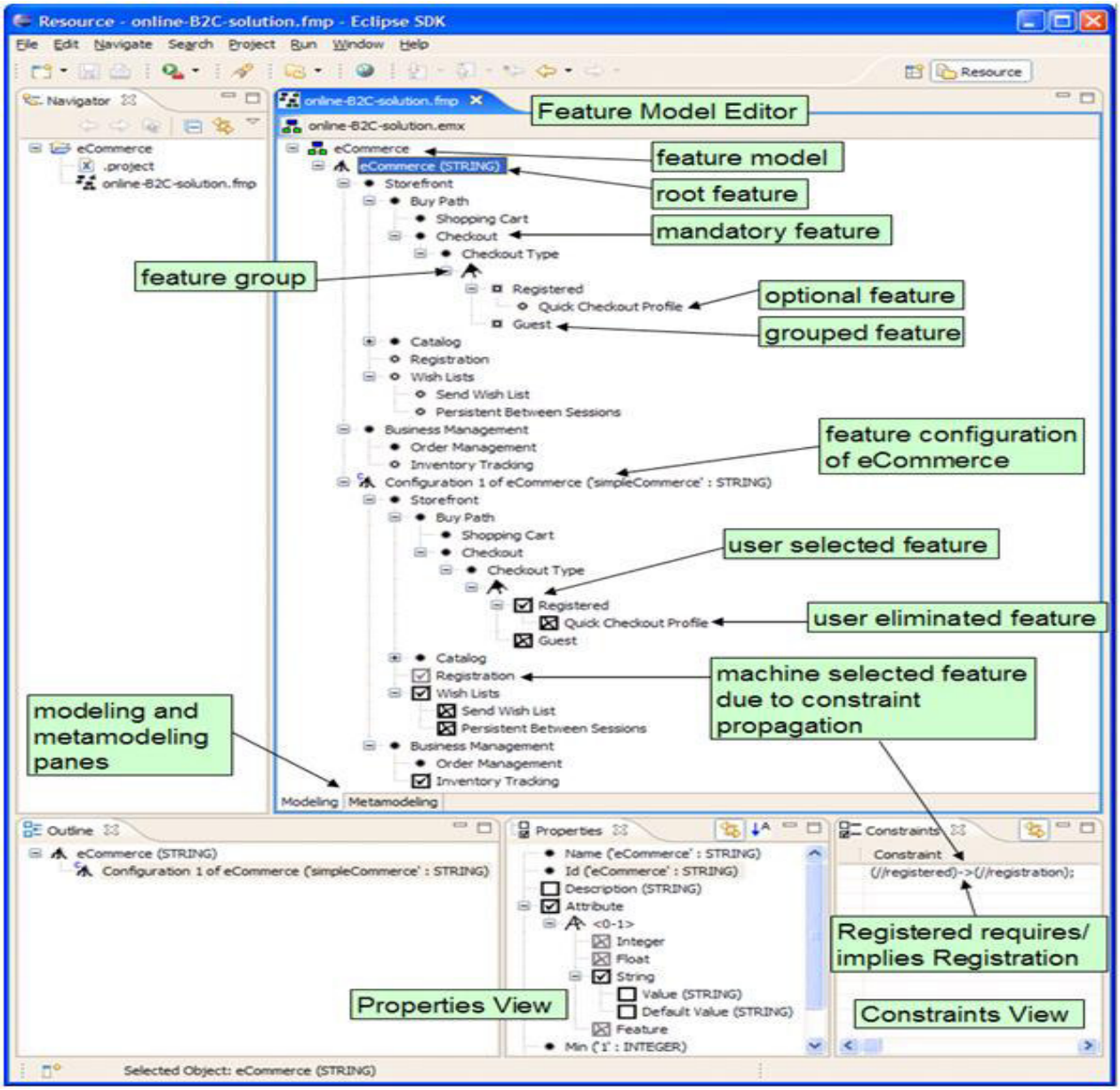


**Figura C-1: Interface do FMP**

[10] (FEATURE MODELING PLUG-IN)

## ANEXO D – PROGRAMAÇÃO ORIENTADA À ASPECTO[11]

Pesquisas têm apontado muitos problemas de programação que nem linguagens procedurais e as orientadas a objetos possuem técnicas suficientemente claras para implementar algumas decisões de projeto. Isto força a implementação destas decisões de projeto a serem espalhadas através do código gerando um código confuso e difícil de desenvolver e manter. Estas decisões de projeto são difíceis de capturar porque elas atravessam as funcionalidades básicas do sistema, ou seja, não é possível separá-la em um componente ou uma classe, pois ela "pertence" a todo sistema.

Neste sentido nasceu a AOP, com o intuito de apresentar técnicas de programação capazes de cobrir estas falhas. A programação orientada a aspectos permite aos programadores separar os interesses comuns do sistema (comportamentos que fogem a típica divisão de responsabilidade, como o logging), introduzindo o conceito de aspectos, os quais encapsulam comportamentos que afetam múltiplas classes.

Esses comportamentos não encaixam naturalmente dentro de um modulo de um programa, ou em vários módulos de programas fortemente relacionados. Os pioneiros da programação orientada a aspectos chamam este tipo de comportamento de crosscutting porque ele atravessa a divisão de responsabilidades dos modelos de programação tradicionais.

Na programação orientada a objetos a unidade natural de modularização é a classe, e um *crosscutting* é uma responsabilidade que está espalhada por muitas classes. Exemplos típico seria o tratamento de erros em contexto sensitivo, otimização de performance, e padrões de projeto.

Trabalhar com código que apontam para responsabilidades que atravessam o sistema gera problemas que resultam da falta de modularidade. Devido a implementação desses comportamentos serem espalhados, os desenvolvedores podem encontrar dificuldades em analisar, implementar e modificar tais comportamentos. Por exemplo, o código de *logging* é entrelaçado ao longo de outros códigos que não estão relacionados com *logging*.

Dependendo da complexidade e do escopo onde é inserido essa responsabilidade, pode resultar em um código confuso que dependendo do local

---

[11] (PACIOS, 2006)

pode ter maiores ou menores implicações. A mudança de uma política de *logging* de uma aplicação pode envolver centenas de modificações, o que aumenta muito a quantidade de código a ser tratada, é o pior esse código vai estar espalhado em vários lugares.

AOP complementa a programação orientada a objeto por facilitar um outro tipo de modularidade, que expande a implementação espalhada de uma responsabilidade dentro de uma simples unidade. Esta unidade é chamada de aspecto, daí o nome programação orientada a aspectos. Os interesses tornam-se fáceis de tratar porque estão concentrados no código de aspectos e afetaram todas as unidades desejadas.

Os aspectos de um sistema podem ser alterados, inseridos ou removidos em tempo de compilação, e frequentemente reusados. Por estar em um único bloco de código a manutenção é muito mais fácil e a complexidade do sistema diminui, facilitando o entendimento do mesmo.

Uma implementação básica de AOP consiste em: uma linguagem de componente para programar os componentes, uma ou mais linguagem de aspectos para programar os aspectos, um *weaver* para combinar as duas linguagens, um programa de componentes e um ou mais programa de aspectos. Existem ferramentas que possuem conceitos de AOP à linguagem Java, como o *AspectJ***.**
Ele utiliza Java como a linguagem para a implementação dos *concerns* individuais, e tem construções para a especificação das regras, que são especificadas em termos de *join points*, *pointcuts* e *advices*, e tudo isto é encapsulado em um *aspect*.

- **Join points:** representam pontos bem definidos na execução de um programa onde um determinado aspecto pode ser aplicado. Em *AspectJ*, *join points* podem ser chamadas de métodos, acessos a membros de uma classe. *Join points* podem conter outros *join points*.

- ***Pointcuts*:** *pointcut* é um agrupamento de *join points* baseando-se em um critério pré-definido.

- ***Advices*:** trechos de código que são executados nos *pointcuts*. Um *advice* contém as alterações que devem ser aplicadas ortogonalmente ao sistema.

- ***Aspects*:** são similares a classes: têm um tipo, podem ser estendidos, podem ser abstratos ou concretos e podem conter campos, métodos e tipos como membros. Mas são diferentes de classes: não têm construtor nem, não podem ser

criados com o operador "*new*", podem conter *pointcuts* e *advices* como membros e podem acessar membros de outros tipos.