Evolução de Componentes Compartilhados por Múltiplas Linhas de Produto de Software

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por Wendel Malta de Assis e aprovada pela Banca Examinadora.

Campinas, 21 de dezembro de 2009.

Cecília Mary Fischer Rubira

Profa. Dra. Cecília Mary Fischer Rubira
Instituto de Computação - UNICAMP
(Orientadora)

Dissertação apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP
Bibliotecária: Maria Fabiana Bezerra Müller – CRB8 / 6162

Assis, Wendel Malta de
As76e Evolução de componentes compartilhados por múltiplas linhas de produto de software/Wendel Malta de Assis -- Campinas, [S.P. : s.n.], 2009.

Orientador : Cecília Mary Fischer Rubira
Dissertação (mestrado) - Universidade Estadual de Campinas, Instituto de Computação.

1.Software (Evolução). 2.Componentes de software. 3.Linhas de produto de software. I. Rubira, Cecília Mary Fischer. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Título em inglês: Evolution of components shared by multiple software product lines

Palavras-chave em inglês (Keywords): 1.Software (Evolution). 2. Software components. 3. Software product lines. 4. Software patterns.

Área de concentração: Engenharia de Software

Titulação: Mestre em Ciência da Computação

Banca examinadora:
Profa. Dra. Cecília Mary Fischer Rubira (IC - UNICAMP)
Profa. Dra. Rosana T. Vaccare Braga (ICMC – USP)
Profa. Dra. Eliane Martins (IC – UNICAMP)

Data da defesa: 21/12/2009

Programa de Pós-Graduação: Mestrado em Ciência da Computação

TERMO DE APROVAÇÃO

Dissertação Defendida e Aprovada em 21 de dezembro de 2009, pela Banca examinadora composta pelas Professoras Doutoras:

Rosana Braga

Profª. Drª. Rosana Teresinha Vaccare Braga
ICMC / USP

Eliane Martins

Profª. Drª. Eliane Martins
IC / UNICAMP

Cecília Mary Fischer Rubira

Profª. Drª. Cecília Mary Fischer Rubira
IC / UNICAMP

Instituto de Computação
Universidade Estadual de Campinas

Evolução de Componentes Compartilhados por Múltiplas Linhas de Produto de Software

Wendel Malta de Assis

Dezembro de 2009

Banca Examinadora:

- Profa. Dra. Cecília Mary Fischer Rubira
  Instituto de Computação - UNICAMP (Orientadora)
- Rosana T. Vaccare Braga
  Instituto de Ciências Matemáticas e de Computação - USP
- Eliane Martins
  Instituto de Computação - UNICAMP
- Ivan Ricarte
  Faculdade de Engenharia Elétrica e de Computação - UNICAMP (Suplente)
- Ariadne M. Rizzoni Carvalho
  Instituto de Computação - UNICAMP (Suplente)

Resumo

O uso de Linhas de Produto de Software é uma prática comum entre as empresas de software, tendo como objetivo promover o desenvolvimento de um conjunto de produtos de software relacionados através da reutilização de um núcleo comum de ativos de software. Dentre estas empresas, podemos mencionar a Motorola, cujo ambiente de desenvolvimento em que múltiplas linhas de produto são mantidas em paralelo serviu de motivação para este trabalho.

Na prática, a complexidade de alguns tipos de negócios apoiados por linhas de produto implica em mudanças na forma como a adoção da abordagem é sugerida pela literatura. Em particular na Motorola, as linhas de produto são baseadas em componentes e a arquitetura da linha de produto possui diversos pontos de variação, onde variantes de componentes representando diversas alternativas de projeto podem ser escolhidas. Além disso, várias linhas de produto são mantidas em paralelo e os componentes e suas variantes podem ser compartilhados entre elas.

Neste contexto, a evolução de componentes é uma tarefa complexa, pois a inclusão de novas características nas variantes dos componentes pode impactar não somente a arquitetura e os ativos de uma única linha de produto, mas também das diversas linhas que as utilizam. A principal contribuição deste trabalho é a documentação de uma família de padrões de evolução de componentes compartilhados por múltiplas linhas de produto de software. Além desta família de padrões, também é apresentado um processo para auxiliar na análise do padrão de evolução a ser adotado para implementar uma determinada requisição de mudança.

Abstract

The Software Product Line approach is becoming widely used by software companies, whose goal is to promote the development of a set of related software products through the reuse of a common core of software assets. Among these companies, we can mention Motorola, whose development environment where multiple software product lines are maintained in parallel served as the motivation for this work.

In practice, the complexity of some types of businesses supported by product lines involves changes in how the adoption of the approach is suggested by the literature. At Motorola, the product lines are based on components and the product line architecture has many variation points, where variants of components representing various design alternatives can be chosen. In addition, several product lines are maintained in parallel and the components and their variants can be shared among them.

In this context, the evolution of components is a complex task, because the inclusion of new features in variants of the components can impact not only the architecture and assets in a single product line but also on many products lines that are using them. The main contribution of this work is the documentation of a family of component evolution patterns that are shared between multiple software product lines. Besides that, a process to assist in analyzing the evolution pattern to be taken to implement a specific change request is presented.

Agradecimentos

Várias pessoas colaboraram direta ou indiretamente para a realização deste trabalho, através de sugestões, idéias, críticas e opiniões. Outras pessoas contribuíram com amizade, carinho e afeto. Agradecer de forma adequada, ou mesmo lembrar de todas essas pessoas, é o que torna a escrita dos agradecimentos mais difícil que redigir a própria dissertação. Entretanto, vou me arriscar e destacar algumas pessoas que foram realmente importantes para este trabalho.

- Um agradecimento muito especial a pessoa mais importante na minha vida, a Bruna. No começo do mestrado, minha namorada. Depois, noiva. Agora, esposa. Obrigado por estar sempre ao meu lado nos momentos alegres, difíceis e, até mesmo, no momento de revisar esta dissertação. O seu apoio foi fundamental para a conclusão deste trabalho.

- Agradeço à minha orientadora, Cecília Rubira, por compreender longos momentos de ausência no mestrado devido as exigências do meu trabalho. Além disso, a objetividade, a exigência de prazos e os seus comentários foram muito importantes para a construção desta dissertação.

- Aos colegas do grupo de pesquisas em engenharia de software, em especial ao Leonardo. As suas dicas de artigos e a ajuda na montagem do estudo de caso foram muito importantes para este trabalho.

- Por fim, mas não menos importante, gostaria de agradecer a minha família, em especial aos meus pais.

Sumário

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

O uso de *Linhas de Produto de Software* (LPS) está se tornando uma prática comum entre as empresas de software, tendo como objetivo promover o desenvolvimento de um conjunto de produtos de software relacionados através da reutilização de um núcleo comum de ativos de software [10]. Dentre esses ativos, a *Arquitetura da Linha de Produto* (ALP) desempenha um papel importante ao apoiar a variabilidade de software com o uso de pontos de variação e ao promover a reutilização em larga escala através da configuração de seus elementos arquiteturais, na forma de componentes e conectores arquiteturais. O termo *Variabilidade de Software* pode ser definido como a capacidade de um ativo de software ser modificado, customizado ou configurado para ser utilizado em um contexto específico. A variabilidade representa decisões de projeto que podem ser realizadas ao longo do processo de desenvolvimento. Um *ponto de variação* é o local do ativo de software em que uma decisão de projeto pode ser tomada e *variantes* são as alternativas de projeto associadas a este ponto. O *Desenvolvimento de Software baseado em Componentes* (DBC) pode ser usado de uma forma complementar com linhas de produtos, pois também possui como objetivo a reutilização de software, visando à redução do custo e do tempo de desenvolvimento de produtos [1]. O desenvolvimento baseado em componentes ocorre através da composição de blocos interoperáveis e reutilizáveis chamados de *componentes de software*, que serão responsáveis por implementar os componentes e conectores de uma arquitetura [33]. Desta forma, a criação de arquiteturas de linhas de produtos baseadas em componentes visa potencializar a reutilização de software.

Na prática, a complexidade de alguns tipos de negócios apoiados por linhas de produtos implica em mudanças na forma como a adoção da abordagem é sugerida pela literatura [37]. Em particular na Motorola, as arquiteturas das linhas são baseadas em componentes e a variabilidade de software é implementada através de diversos pontos de variação presentes na sua arquitetura, onde variantes de componentes representando diversas al-

ternativas de projeto podem ser escolhidas. Assim, a variante é um componente que materializa um ponto de variação na arquitetura da linha de produto. Quando uma nova linha de produtos é criada, o gerente responsável pela linha define o conjunto de características da linha e divulga-o para os gerentes de cada um dos componentes, responsáveis por avaliar o impacto das mudanças de características nas diversas variantes de cada componente. Estas mudanças podem ocasionar desde pequenas alterações na implementação até alterações nas interfaces dos componentes. O gerente de um componente, após a análise de impacto das mudanças, pode decidir criar uma nova variante do componente para atender à demanda da nova linha, dado que a evolução de variantes já existentes pode afetar de forma negativa as linhas em operação. Nesse contexto, onde variantes de componentes são compartilhadas por múltiplas arquiteturas de linhas de produtos, a evolução dos ativos controlada pelos gerentes dos componentes torna-se uma tarefa complexa, pois a inclusão de novas características nas variantes dos componentes pode impactar não somente a arquitetura e os ativos de uma única linha de produto, mas também das diversas linhas que as utilizam. Este cenário também ocorre em outras empresas, como *Philips Eletronics* [37] e *Nokia Corporation* [28].

Na literatura, existem vários trabalhos voltados ao estabelecimento de linhas de produto de software em empresas, mas poucos tratam do problema da evolução de linhas de produto em operação [32]. Além disso, a maior parte destas pesquisas [22] [25] [32], analisam apenas os impactos da evolução no contexto de uma única linha de produto. A evolução de um componente quando ele é compartilhado por múltiplas linhas de produto é bem menos estudado[37].

Os problemas relacionados com a evolução de componentes compartilhados por múltiplas arquiteturas de linhas de produto são recorrentes. Por isso, eles podem ser categorizados através de padrões de evolução. Por exemplo, a adição de uma nova característica em um componente compartilhado por duas ou mais linhas de produto pode representar um risco desnecessário sob o ponto de vista das linhas que não requisitaram esta alteração. Desta forma, a criação de uma nova variante para incluir a mudança solicitada é uma possível solução para não afetar outras linhas em operação. Em outros casos, uma característica implementada em uma variante de um componente pode ser requisitada por outra linha de produto que não a implementava, dado que essa característica pode trazer uma vantagem competitiva sob o ponto de vista do mercado. Além disso, a manutenção de múltiplas variantes de um componente em paralelo sugere que a correção de erros em uma variante deve ser propagada para as demais.

A principal contribuição deste trabalho é a criação de uma família de padrões de evolu-

ção que documentam diversos cenários de evolução para o contexto onde várias variantes de componentes de software são compartilhadas por múltiplas arquiteturas de linhas de produto. Os principais padrões identificados foram: (i) Criar ou manter múltiplas variantes de um componente em paralelo, (ii) Criar ou manter múltiplas variantes temporárias de um componente, (iii) Propagar correção de erros entre variantes de um componente, (iv) Incluir característica de uma variante em outra do mesmo componente e (iv) Convergir várias variantes de um componente criando uma nova variante. A identificação destes padrões ocorreu a partir da análise das requisições de mudanças realizadas nas linhas de produto mantidas pela Motorola nos últimos cinco anos. Para cada padrão, é discutido o seu uso, suas vantagens e limitações, seus possíveis riscos e impactos nos artefatos associados ao componente. O uso da família de padrões de evolução estabelece uma linguagem comum para a evolução de componentes compartilhados. Assim, quando requisições de mudanças forem solicitadas pelos gerentes de linhas de produto, os gerentes de componentes poderão consultar um catálogo que documenta possíveis cenários a serem adotados e os seus riscos associados. Dessa forma, a evolução das linhas poderá ser realizada de uma forma mais controlada e sistemática.

A escolha do padrão de evolução a ser adotado ao se implementar uma requisição de mudança de característica será apoiada por um processo de análise simplificado. Além disso, uma ferramenta para auxiliar na escolha da variante de um componente a ser utilizada por uma linha de produto é apresentada.

1.1 Motivação e Problema

A definição do escopo de uma linha de produto corresponde a primeira etapa no processo de implantação da abordagem de linha de produto de software. Nesta fase, deve-se identificar as entidades com as quais os produtos irão interagir e estabelecer os pontos comuns e as variabilidades que devem ser implementadas. Isto é, deve ser definido o que está "dentro" e "fora" da linha de produto. A especificação do escopo, a análise de domínio e o levantamento de requisitos são as principais atividades que, em conjunto, são responsáveis pela especificação do contexto em que a linha de produto será desenvolvida[26].

Entretanto, uma linha de produto de software, assim como qualquer sistema de software, evolui ao longo do tempo. Essa evolução ocorre através de alterações das características iniciais dos produtos da família, sejam elas representadas por requisitos funcionais ou não-funcionais. Novas características podem surgir a partir de diversas fontes, como: clientes que utilizam os produtos, novas tecnologias que permitem o oferecimento de novos serviços, necessidades futuras identificadas pelas organizações, novos padrões, correções

de defeitos, entre várias outras [24].

No contexto deste trabalho, as requisições de mudanças para um componente ocorrem a partir de várias linhas de produtos que o compartilham. A Figura 1.1 ilustra este contexto através da representação de um conjunto de arquiteturas de linha de produto (`ALP A`, `ALP B` e `ALP C`) e um núcleo de ativos comum entre as três linhas, onde estão armazenados os componentes compartilhados entre elas (`Componente X`, `Componente Y` e `Componente Z`). Os componentes não compartilhados entre as linhas fazem parte dos ativos centrais de cada linha de produto, como no caso dos componentes `Componente A_1` e `Componente A_2` pertencentes a `ALP A`. Quando alterações no modelo de características são requisitadas e a sua implementação impacta os componentes comuns entre as linhas, novas variantes podem ser criadas para apoiar essas mudanças. Esta abordagem pode ser necessária para minimizar o risco associado às alterações nas linhas de produtos que não solicitaram as mudanças. Assim, torna-se necessário um melhor conhecimento dos padrões de evolução associados a este contexto, de forma a reduzir os riscos e garantir que a evolução ocorra de forma controlada e sistemática.

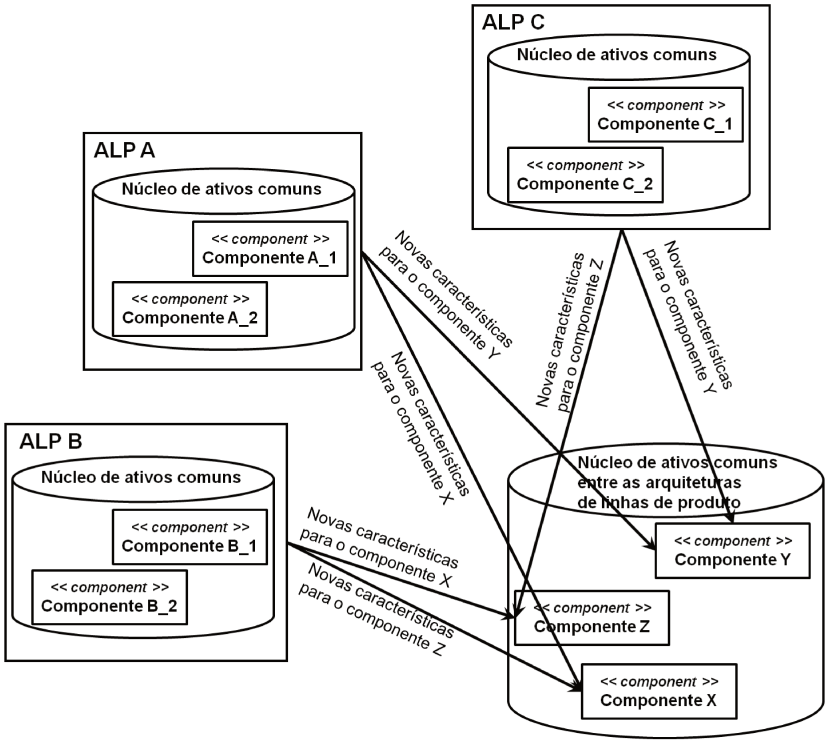

Figura 1.1: Requisições de mudanças realizadas a componentes compartilhados por múltiplas LPSs.

O foco deste trabalho é na análise dos cenários de evolução que podem ocorrer em empresas que mantêm várias linhas de produto em paralelo, onde um conjunto de componentes de software pode ser compartilhado entre as arquiteturas de diversas linhas. O objetivo é identificar cenários de evolução que acontecem de forma recorrente, suas vantagens e limitações, seus possíveis riscos e impactos nos artefatos associados ao componente. Uma vez identificados esses cenários, um processo para auxiliar na análise de qual padrão de evolução deve ser utilizado ao se implementar uma requisição de mudança é apresentado. Além disso, o processo de escolha da variante de um componente a ser utilizada para resolver os pontos de variação arquiteturais existentes também é definido.

1.2 Solução Proposta

Neste trabalho é apresentada uma família de padrões de evolução para o contexto em que um componente de software é compartilhado por múltiplas linhas de produto de software. Os padrões descritos são:

- *Criar ou manter múltiplas variantes de um componente em paralelo*: uma única variante de um componente pode dar origem a uma ou mais variantes que serão mantidas em paralelo à variante original. Essas novas variantes possuem ciclo de vida próprio e poderão dar origem a outras.

- *Criar ou manter múltiplas variantes temporárias de um componente*: uma variante de referência para um componente é definida e novas variantes são temporariamente derivadas para atender a um demanda específica. A variante de referência sempre representará um superconjunto das características de todas as suas variantes derivadas.

- *Propagar correção de erros entre variantes de um componente*: quando uma correção de erro é realizada em uma determinada variante de um componente, deve-se analisar a necessidade de propagação da correção para as demais variantes existentes.

- *Convergir várias variantes de um componente criando uma nova variante*: uma variante de um componente é criada a partir de duas ou mais variantes existentes.

- *Incluir característica de uma variante em outra do mesmo componente*: uma linha de produto que utiliza uma determinada variante de um componente faz uma requisição de uma característica disponível apenas em outra variante do mesmo componente.

A elaboração desta família de padrões permite estabelecer uma linguagem comum para a evolução de componentes no contexto de múltiplas linhas de produto de software. Além

disso, contribui para que a evolução ocorra de forma controlada e sistemática. Para cada padrão, são apresentadas as informações do seu contexto de uso, as vantagens de sua utilização, limitações e os impactos em cada artefato do componente (requisitos, arquitetura interna e projeto). Este detalhamento permite identificar com maior facilidade o risco associado à adoção de cada padrão de evolução.

A família de padrões de evolução foi identificada através de estudos realizados em linhas de produto de software para celulares mantidas pela *Motorola Inc.* Após a fase de identificação e especificação dos padrões, um estudo de caso foi realizado usando uma linha de produto para uma aplicação de manipulação de fotos, vídeos e músicas em telefones celulares, denominada *MobileMedia*. Esta linha de produto foi desenvolvida inicialmente por Trevor Young[38] e estendida por Eduardo Figueiredo et al.[15], com o propósito de ser utilizada para fins educacionais. O objetivo deste estudo de caso foi permitir uma maior compreensão das características de cada cenário de evolução e uma melhor análise do impacto que cada um exerce sobre os artefatos de software associados a um componente.

Para apoiar a escolha da variante de um componente a ser utilizada por uma arquitetura de linha de produto, uma tabela responsável por mapear as variantes dos componentes as características existentes é apresentada.

A escolha do padrão de evolução a ser utilizado, quando uma requisição de mudança de característica é realizada, é apoiada por um processo simplificado que apresenta um conjunto de atividades associadas aos responsáveis pela linha de produto e pelos componentes existentes.

A Figura 1.2 ilustra as três maiores contribuições deste trabalho:

- Família de padrões de evolução;
- Tabela de mapeamento que relaciona as características existentes às variantes disponíveis de um componente e o processo para a sua utilização ao se instanciar a arquitetura da linha de produto;
- Processo para auxiliar na análise do padrão de evolução a ser adotado para se implementar uma requisição de mudança de característica.

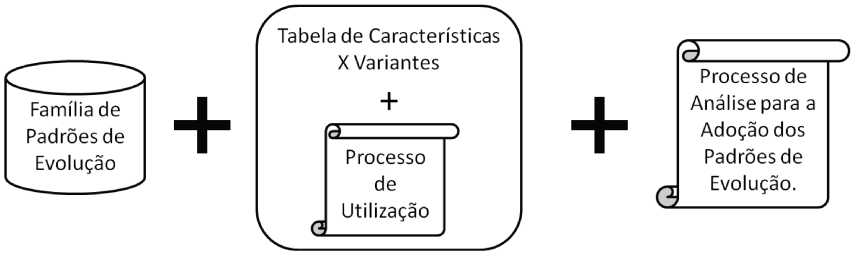

Figura 1.2: Conjunto das soluções propostas.

1.3 Trabalhos Correlatos e Comparação com a Solução Proposta

Esta seção apresenta os trabalhos correlatos na área de evolução de componentes em linhas de produto de software e realiza uma análise comparativa entre esses trabalhos e a família de padrões de evolução elaborada nesta dissertação.

`Svahnberg et al.` [32], baseados em dois estudos de caso, apresentam categorias de evolução dos requisitos, da arquitetura da linha de produto e dos componentes que a constituem, estabelecendo o relacionamento entre elas. Assim, criaram uma taxonomia que permite uma melhor compreensão do processo de evolução e o impacto causado por um novo requisito em toda a linha de produto. Entretanto, o número de estudos de casos não é suficiente para validar as categorias encontradas e não foram apresentadas diretrizes para auxiliar na implementação de cada tipo de evolução. Além disso, as características da evolução em componentes compartilhados por diversas linhas de produto não foram consideradas. A família de padrões de evolução apresentada nesta dissertação corresponde a uma extensão deste trabalho para o contexto de múltiplas linhas de produto que mantêm componentes compartilhados entre elas.

`Bosch et al.` [9] [8], baseados em estudos de caso, enumeram algumas situações que podem levar empresas a manter múltiplas variantes (implementações) de um componente em paralelo. Assim, este trabalho faz parte da motivação para esta dissertação. A principal contribuição dos autores está na enumeração das situações que podem levar a este tipo de evolução e as suas causas primárias. Entretanto, não foram apresentadas soluções para evitar ou gerenciar de forma controlada este cenário.

`Mohan et al.` [25], baseados em um estudo de caso, identificaram alguns problemas que podem ocorrer quando a inserção ou alteração de características ocorre diretamente nos produtos de uma linha de produto, para evitar o impacto nos demais produtos. Esta

abordagem se assemelha a criação de variantes de um componente. Entretanto, o contexto abordado é o de uma única linha de produto e o time responsável pelo componente não é informado das alterações realizadas pelo time de produto. Como resultado, os autores obtiveram alguns padrões de mudança e definiram recomendações para tratar esses padrões. Entretanto, as recomendações apresentadas são muito específicas para serem aplicadas diretamente no contexto de múltiplas linhas de produto de software.

`Riva et al.` [28] apresentam uma arquitetura que utiliza o conceito de camadas para apoiar diversas linhas de produto com uma única arquitetura de referência. Neste cenário, os componentes presentes na arquitetura de referência são reutilizados ao longo de várias linhas de produto. Assim, esse trabalho faz parte da motivação para esta dissertação. Entretanto, o objetivo dos autores é analisar o impacto causado pela evolução na arquitetura da linha de produto. Para isso, eles apresentam os resultados da aplicação de dois métodos (Avaliação da Arquitetura e Reconstrução da Arquitetura) que, em conjunto, tem o objetivo de garantir a consistência da arquitetura ao longo dos ciclos de inserção de novos requisitos. Os autores também apresentam os estágios pelos quais uma organização pode passar ao evoluir para uma abordagem de linha de produto.

`Taborda` [34] [35] propõe a utilização de matrizes para representar as dependências entre produtos e componentes, como forma de apoiar o planejamento e o gerenciamento da evolução de múltiplas linhas de produto que compartilham componentes. A família de padrões de evolução apresentada nesta dissertação corresponde a uma extensão desse trabalho do ponto de vista da gerência de diversas variantes de um componente. Além disso, a técnica de gerenciamento apresentada é utilizada neste trabalho para auxiliar na escolha da variante de um componente a ser utilizada por uma linha de produto.

Os seguintes itens não foram abordados pelos trabalhos relacionados:

- Diretrizes para auxiliar a gerenciar a evolução de múltiplas variantes de componentes;
- O conceito de múltiplas linhas de produto de software mantidas em paralelo não foi formalmente apresentado;
- Evolução de componentes compartilhados por múltiplas linhas de produto;
- Possibilidade de um componente compartilhado por múltiplas linhas de produto possuir várias variantes;
- Metodologia para relacionar as diversas variantes de um componente às características disponibilizadas.

1.4 Organização deste Documento

O restante deste documento está dividido em mais cinco capítulos, organizados da seguinte forma:

- **Capítulo 2 - Fundamentos de Reutilização e Evolução de Software:** apresenta os conceitos e termos importantes para a contextualização deste trabalho. São abordados os assuntos: Reutilização de Software, Desenvolvimento Baseado em Componentes, Arquitetura de Software, Linha de Produto de Software, O Processo de Evolução de Software, Evolução em Linha de Produto de Software e Gerência de Configuração.

- **Capítulo 3 - Evolução de Componentes Compartilhados por Múltiplas Linhas de Produto de Software:** apresenta a motivação para este trabalho dentro do contexto de desenvolvimento da Motorola e uma visão geral dos padrões de evolução identificados através de um exemplo ilustrativo. Além disso, um processo para auxiliar na análise do padrão de evolução a ser adotado quando uma requisição de mudança é realizada e na escolha da variante de um componente a ser utilizada é apresentado.

- **Capítulo 4 - A Família de Padrões de Evolução:** apresenta a família de padrões de evolução para componentes compartilhados por múltiplas linhas de produto de software em maiores detalhes.

- **Capítulo 5 - Estudo de Caso: Linha de Produto MobileMedia:** apresenta o estudo de caso realizado com a linha de produto para fins educacionais conhecida por *MobileMedia*.

- **Capítulo 6 - Conclusões:** apresenta as conclusões desse trabalho, sintetizando suas contribuições, identificando limitações e sugerindo trabalhos futuros.

Capítulo 2

Fundamentos de Reutilização e Evolução de Software

Este capítulo estabelece a terminologia e descreve os fundamentos teóricos importantes para a compreensão deste trabalho. A Seção 2.1 apresenta o conceito de reutilização de software em conjunto com as tecnologias que possibilitam a sua aplicação, como: Componentes de Software, Variabilidade de Software, Linhas de Produto de Software e Arquitetura de Software. A Seção 2.2 apresenta os princípios relacionados à evolução de software e as suas implicações no contexto de linhas de produto de software.

2.1 Reutilização de Software

A *reutilização de software* ocorre quando um artefato existente é utilizado para o desenvolvimento ou manutenção de um produto alvo. O *escopo da reutilização* pode ser muito pequeno, como na reutilização de um artefato de uma versão do produto para a outra, ou mais amplo, como quando o mesmo artefato é utilizado por diferentes produtos em uma linha de produto ou entre produtos pertencentes a diferentes linhas de produtos. A *granularidade da reutilização* também pode variar da simples reutilização de código e documentos de projeto até requisitos, casos de teste, planejamento, cronogramas ou o produto como um todo. Além disso, a reutilização pode ocorrer em qualquer fase do ciclo de vida de um produto. Na *reutilização caixa-preta*, o artefato é utilizado sem mudanças. No caso da *reutilização caixa-branca*, o artefato é modificado para se adaptar ao produto alvo [36].

As próximas seções apresentam uma breve descrição da tecnologia de componentes (Seção 2.1.1), variabilidade de software (Seção 2.1.2), linhas de produto de software (Seção 2.1.3) e arquitetura de software (Seção 2.1.4), que são ferramentas muito utilizadas para

a aplicação das práticas de reutilização de software.

2.1.1 Componentes de Software e Desenvolvimento Baseado em Componentes

Uma definição bem difundida do termo componente de software foi apresentada por `Szyperski` [33]. Ele define *componente de software* como sendo uma unidade de composição com interfaces e dependências de contexto bem definidas. Devido a estas características, um componente de software pode ser desenvolvido de forma independente e está sujeito à composição por terceiros.

As regras para a criação, composição e comunicação de componentes são definidas por *modelos de componentes* e realizadas por *arcabouços de componentes*[1] [31]. Uma *interface* identifica um ponto de interação entre um componente e o seu ambiente. Um componente possui *interfaces providas*, através das quais ele declara os serviços oferecidos ao ambiente e *interfaces requeridas*, pelas quais ele declara os serviços do ambiente dos quais depende para funcionar [3].

Um componente de software pode ser atômico ou composto por vários outros componentes, chamados de *subcomponentes*. Um *componente abstrato* é uma descrição do seu comportamento observável que pode ser utilizado como elemento de projeto para o desenvolvimento da arquitetura de um sistema, de forma independente de qualquer implementação. Um *componente concreto* é uma implementação do comportamento especificado pelo componente abstrato e pode ser utilizado como item de configuração para compor diferentes sistemas de software executáveis [18].

Um componente é responsável por implementar pelo menos uma característica requisitada pelo sistema em que ele será utilizado. Uma *característica* é uma propriedade do sistema que é relevante para alguma parte interessada do projeto[13]. Entretanto, uma característica também pode ser implementada parcialmente por dois ou mais componentes e um componente pode implementar mais de uma característica. Nesta dissertação, um vetor de características será associado a cada componente. Este vetor será representado por: *[<Característica1>,<Característica2>,...,<CaracterísticaN>]*, onde:

- *<CaracterísticaX>* identifica uma característica implementada por um componente, quer ela seja parcialmente implementada ou não.
- *[*, *,* e *]* são caracteres separadores.

[1] do inglês, *frameworks*

A Figura 2.1 ilustra a representação de um componente, com suas interfaces e seu vetor de características. O estereótipo *component* será utilizado para indicar que o ativo ilustrado é um componente de software. Neste exemplo, o componente chama-se `Componente X` e o seu vetor de características é `[A,B,C]`, ou seja, as características `A`, `B` e `C` são implementadas parcialmente ou de forma completa pelo componente. A representação do vetor de características não é obrigatória.

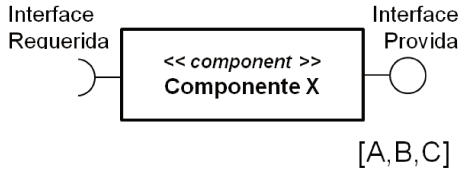

Figura 2.1: Representação gráfica de um Componente de Software.

O *Desenvolvimento Baseado em Componentes* (DBC) é uma metodologia voltada para a construção rápida de sistemas a partir de componentes, onde: (1) os componentes possuem propriedades bem definidas e (2) essas propriedades oferecem a base para prever as características dos sistemas construídos a partir dos componentes [3]. O DBC permite que um sistema seja construído através da composição de componentes de software previamente especificados, construídos e testados, o que possibilita um ganho de produtividade e qualidade. [33].

2.1.2 Variabilidade de Software

O termo *Variabilidade de Software* pode ser definido como a capacidade de um ativo de software ser modificado, customizado ou configurado para ser utilizado em um contexto específico. A variabilidade representa decisões de projeto que podem ser realizadas ao longo do processo de desenvolvimento. Um *ponto de variação* é o local do ativo de software em que uma decisão de projeto pode ser tomada e *variantes* são as alternativas de projeto associadas a este ponto.

2.1.3 Linha de Produto de Software

A abordagem de *Linha de Produto de Software* (LPS) consiste do desenvolvimento de um conjunto de produtos de software, com alto grau de similaridade entre si e que atendem às necessidades específicas de um segmento de mercado ou missão, de forma prescritiva a partir de um conjunto de ativos centrais [10].

De certa forma, uma linha de produto de software utiliza conceitos e técnicas do paradigma de desenvolvimento baseado em componentes. Entretanto, a principal diferença entre estas abordagens está no fato de que os componentes e as diferentes características existentes entre os produtos de uma LPS já foram previamente especificados. Com isso, a reutilização ocorre de forma planejada e prescritiva, a partir de componentes mantidos em um repositório comum aos produtos da linha de produto. Em uma típica definição de desenvolvimento baseado em componentes, a construção dos produtos ocorre através da seleção de componentes em uma biblioteca interna ou no mercado. Neste caso, o reuso é oportunístico e arbitrário.

Os círculos da Figura 2.2[27] representam as três atividades essenciais para a implantação e manutenção de uma linha de produto:

- O *Desenvolvimento de Ativos Centrais* é a atividade contínua de criação e manutenção dos ativos centrais que serão reutilizados entre os produtos da família de produtos. Esta atividade está relacionada a outra área de pesquisa conhecida por *Engenharia de Domínio*, que consiste nas atividades de coletar, organizar e armazenar experiências anteriores na construção de aplicações em um domínio específico na forma de artefatos reutilizáveis. A Engenharia de Domínio consiste nas fases de Análise de Domínio (definição de escopo e modelagem do domínio), Projeto de Domínio (especificação da arquitetura) e Implementação de Domínio (implementação da arquitetura e dos componentes do domínio). Entretanto, a principal diferença entre Linhas de Produto de Software e Engenharia de Domínio está no foco em produtos ao invés de um domínio específico. Os produtos podem incluir vários domínios de aplicação e normalmente cobrem somente uma parte do domínio completo.[12]

- O *Desenvolvimento de Produtos* é a atividade de construção dos produtos através da utilização dos ativos centrais. Esta atividade está relacionada a outra área de pesquisa conhecida por *Engenharia de Aplicação*, que é o processo de construção de sistemas a partir da seleção de um subconjunto de artefatos reutilizáveis que foram desenvolvidos pela Engenharia de Domínio.[12]

- O *Gerenciamento* representa as atividades de gerenciamento técnico e organizacional que apoiam a linha de produto.

A sobreposição entre os círculos sugere que as atividades ocorrem em paralelo e são relacionadas. Além disso, não existe nenhuma ordem de execução entre elas. Com isso, os ativos centrais podem ser desenvolvidos baseado em especificações de produtos alvos da linha de produto, a partir de um conjunto de produtos existentes ou de qualquer combinação dessas abordagens. Em resumo, o desenvolvimento de uma linha de produto consiste

Figura 2.2: Atividades essenciais em uma Linha de Produto de Software.

no desenvolvimento de ativos reutilizáveis e produtos que os utilizam, com o suporte das atividades de gerenciamento.

As próximas subseções apresentam os padrões que auxiliam na criação e manutenção de linhas de produto (Seção 2.1.3) e a técnica mais utilizada para a modelagem das características comuns e variáveis em uma linha de produto (Seção 2.1.3).

Padrões de Linha de Produto

`Clements et al.`[10] descreveram um conjunto de vinte e nove áreas de conhecimento[2] que uma organização deve possuir para ser capaz de trabalhar com linhas de produto de software. Além disso, elaboraram diversos *padrões de linha de produto*[3] que coordenam a aplicação das áreas de conhecimento para auxiliar nas várias fases da implantação e manutenção de linhas de produto. Os padrões apresentados foram:

- *Currículo*[4]: agrupa as vinte e nove áreas de conhecimento em três categorias, que representam o conjunto de competências necessárias para implementá-las;
- *Cobertura Essencial*[5]: relaciona as áreas de conhecimento e as atividades essenciais de uma linha de produto (Seção 2.1.3);
- *Cada Ativo*[6]: apresenta as áreas de conhecimento necessárias para se implementar

[2]do inglês, *practice areas*
[3]do inglês, *software product line practice patterns*
[4]do inglês, *Curriculum*
[5]do inglês, *Essentials Coverage*
[6]do inglês, *Each Asset*

cada ativo de uma linha de produto de software;

- *O que Produzir*[7]: auxilia a identificar quais produtos serão produzidos.
- *Partes dos Produtos*[8]: auxilia a definir quais ativos devem ser produzidos.
- *Linha de Montagem*[9]: áreas de conhecimento que permitem iniciar o processo de produção de uma linha de produto.
- *Monitorar*[10]: apresenta formas de se monitorar o estado de uma linha de produto.
- *Construir Produto*[11]: apresenta as áreas de conhecimento que devem ser executadas para se produzir um produto.
- *Iniciar a Implantação de Linha de Produto*[12]: representa as áreas de conhecimento necessárias para se iniciar o processo de implantação de uma linha de produto.
- *Manter uma Linha de Produto*[13]: representa as áreas de conhecimento necessárias para se manter uma linha de produto em atividade.
- *Processos*[14]: indica as áreas de conhecimento relacionadas à definição de processos.
- *Fábrica*[15]: coordena a aplicação de todos os padrões com o objetivo de estabelecer e iniciar a produção de uma linha de produto de software.

`McGregor`[24] apresenta também o padrão *Evoluir Cada Ativo*[16], onde o ativo previamente construído seguindo o padrão *Cada Ativo*, é modificado.

Modelagem de Características

A *Modelagem de Características* é uma técnica muito utilizada para a modelagem dos aspectos comuns e variáveis entre os produtos de uma linha de produto de software. Ela faz parte do método de análise de domínio conhecido por FODA (*Feature-Oriented Domain Analysis*) [20]. As características a serem modeladas podem ser: serviços (redirecionamento de chamadas), operações (enviar mensagem de texto), propriedades não-funcionais

[7] do inglês, *What to Build*
[8] do inglês, *Product Parts*
[9] do inglês, *Assembly Line*
[10] do inglês, *Monitor*
[11] do inglês, *Product Builder*
[12] do inglês, *Cold Start*
[13] do inglês, *In Motion*
[14] do inglês, *Process*
[15] do inglês, *Factory*
[16] do inglês, *Evolve Each Asset*

(desempenho) e tecnologias (algoritmos de tratamento de erros na transmissão de rádio). De acordo com esta técnica, as características de um sistema de software são documentadas em um *modelo de características*, constituído por:

- *um diagrama de características*: decomposição hierárquica das características, indicando quais são mandatórias, alternativas ou opcionais. As características comuns (mandatórias) são aquelas que necessariamente têm que estar presentes em qualquer instanciação do produto, as opcionais são aquelas selecionáveis, sem nenhuma restrição, nas configurações do produto e as alternativas indicam que não mais que uma característica pode ser selecionada para um produto.

- *um conjunto de regras de composição*: indicações de quais combinações de características são válidas.

A Figura 2.3 representa o modelo de características de um caixa eletrônico de auto-atendimento. A característica raiz, `Caixa Eletrônico`, é composta por duas características comuns: `Identificação do Usuário` e `Emissão de Extrato`. A característica `Identificação de Usuário` consiste em duas características alternativas `Toque de Tela` e `Leitora de Cartão`, ou seja, somente uma delas pode ser selecionada. A característica `Emissão de Extrato` consiste em duas características ou-inclusivas: a característica comum `Visor` e a característica opcional `Impressora`. Isto quer dizer que as opções possíveis de saída de extrato são somente visor, ou visor e impressora.

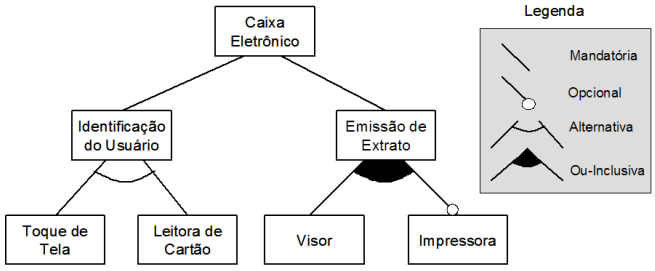

Figura 2.3: Modelo de características de um Caixa Eletrônico.

O vetor de características de um componente também pode ser apresentado graficamente utilizando o formato proposto pelo diagrama de características. A Figura 2.4 estende a Figura 2.1 para ilustrar o componente `Componente X` e o seu vetor de características `[A,B,C]` representado como uma árvore de características. Neste exemplo, as características `A` e `B` são mandatórias e a característica `C` é opcional.

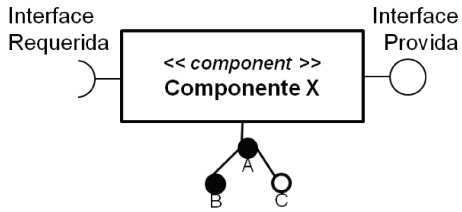

Figura 2.4: Representação gráfica de um Componente de Software, onde o vetor de características é apresentado no formato de árvore de características.

2.1.4 Arquitetura de Software

A *arquitetura de software* de um programa ou sistema computacional é a estrutura ou estruturas do sistema, compreendendo: os elementos de software, as propriedades externamente visíveis desses elementos e o relacionamento existente entre eles [4]. Ao utilizar o termo elementos de software, a definição não se prende a nenhuma tecnologia específica para a implementação desses elementos.

A arquitetura abstrai o sistema através da definição de seus elementos de software e da forma como eles interagem entre si. Com isso, os detalhes de implementação dos elementos não fazem parte da arquitetura, apenas as informações sobre os serviços disponibilizados por eles, bem como características de performance, tratamento de exceções, entre outras propriedades. A decomposição e organização de um sistema de software em subsistemas menores é uma abordagem comumente utilizada para se lidar com a complexidade e o tamanho dos sistemas atuais. [4]

Arquiteturas de software são importantes, pois elas representam uma abstração comum do sistema que pode ser utilizada na comunicação entre os interessados[17] do projeto. Além disso, elas documentam as decisões de projeto, que podem ser reutilizadas na construção de soluções para sistemas semelhantes [4].

Uma *Arquitetura de Software Baseada em Componentes* define os elementos de software como sendo componentes e conectores arquiteturais. Um *componente arquitetural* é responsável por implementar a funcionalidade de uma parte do sistema. Eles podem ser definidos em diferentes níveis de granularidade, provendo desde funcionalidades básicas, como operações matemáticas, até um conjunto amplo de funcionalidades. Um *conector arquitetural* tem como principal responsabilidade mediar a interação entre elementos arquiteturais. Além disso, ele também pode ser responsável por uma parcela dos aspectos

[17] do inglês, *stakeholder*

de qualidade do sistema, tais como distribuição e segurança. Uma determinada organização de componentes e conectores arquiteturais em um sistema é denominada *configuração arquitetural* [30].

A Figura 2.5 ilustra uma arquitetura de software baseada em componentes para celulares. Os componentes são `Mensagem`, `Mensagem Instantânea` e `Agenda`. Os conectores não são mostrados explicitamente, mas consistem das ligações entre as interfaces providas e requeridas dos componentes.

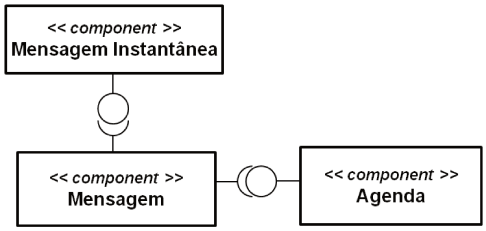

Figura 2.5: Uma Arquitetura de Software Baseada em Componentes.

Na arquitetura da Figura 2.5, supondo que os componentes `Mensagem`, `Mensagem Instantânea` e `Agenda` implementam os vetores de características `[Enviar Mensagem, Receber Mensagem]`, `[Enviar Mensagem Instantânea]` e `[Recuperar Contato]`, pode-se dizer que o conjunto desses vetores de características corresponde às características disponibilizadas pelos sistemas de software criados a partir desta arquitetura.

Na abordagem de linha de produto de software, a arquitetura de software é muito importante e faz parte dos ativos centrais da linha de produtos. Uma *Arquitetura de Linha de Produto* (também conhecida como *Arquitetura de Referência* ou *Arquitetura de Domínio*) é uma arquitetura de software que apoia uma família de produtos, refletindo as características comuns e as variabilidades entre os vários produtos [5]. A arquitetura de linha de produto deve se preocupar em identificar e fornecer mecanismos para exercitar um conjunto de variações explicitamente permitido, pois estas variações darão origem aos produtos apoiados pela linha de produto [10].

Nesta dissertação, a arquitetura de linha de produto permite que a variabilidade em nível arquitetural seja implementada através de pontos de variação onde é possível escolher entre diferentes variantes de um mesmo componente. Desta forma, o estereótipo *variant component* será utilizado para representar um ponto de variação associado a um componente na arquitetura, onde uma variante deste componente será escolhida de acordo com as características exigidas por um determinado produto. A Figura 2.6 altera a arqui-

tetura de componentes ilustrada na Figura 2.5, de forma a tornar o componente `Mensagem` um ponto de variação na arquitetura.

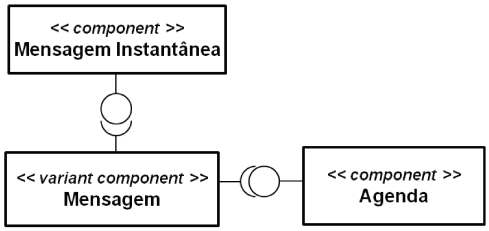

Figura 2.6: Uma Arquitetura de Linha de Produto.

`Bachmann et al.` [2] apresentam outra forma de representação onde o ponto de variação e as variantes associadas a ele são apresentadas simultaneamente. No exemplo ilustrado pela Figura 2.7, as variantes `Mensagem_1` e `Mensagem_2` do componente `Mensagem` são apresentadas. Ao longo deste trabalho, ambas representações são utilizadas.

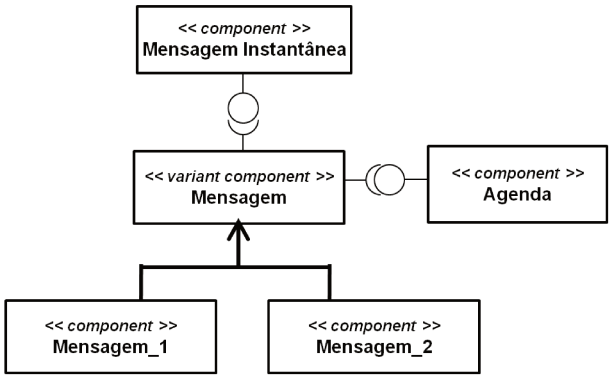

Figura 2.7: Uma Arquitetura de Linha de Produto, onde pontos de variação e variantes são ilustrados simultaneamente.

2.2 Evolução de Software

Após ser colocado em produção, um sistema de software pode se tornar obsoleto caso não sofra mudanças ao longo do seu ciclo de vida, podendo ser necessário adaptá-lo a novas plataformas e realizar alterações em suas propriedades não-funcionais. Além disso, a utilização do sistema faz com que novos requisitos e modificações nos requisitos existentes sejam propostos e erros sejam encontrados. Por isso, o desenvolvimento de software é uma

atividade que não acaba quando um sistema é entregue e colocado em funcionamento, mas continua durante todo o seu ciclo de vida [30].

A *evolução de software* não se limita apenas a analisar as mudanças ocorridas após o software ser entregue ao cliente. A preocupação com a evolução deve ocorrer desde o desenvolvimento inicial, onde ferramentas e tecnologias que auxiliem na criação de uma arquitetura de software mais flexível devem ser utilizadas, tornando mais fácil a integração de mudanças futuras. Além disso, uma documentação adequada do sistema e das decisões de projetos realizadas deve ser disponibilizada. Idealmente, a equipe responsável pelo desenvolvimento inicial de um sistema deveria ser responsável por suas atualizações. Entretanto, sistemas de software personalizados podem ser desenvolvidos externamente e a evolução ser de responsabilidade do cliente ou de uma empresa externa [30].

A definição para o termo evolução de software ainda não possui um consenso na literatura. Além disso, o termo *manutenção de software* é utilizado por alguns autores como um sinônimo, uma variação ou uma fase da evolução de software [6]. Neste trabalho, o termo evolução de software será adotado e definido como sendo a necessidade contínua de se alterar um sistema de software para apoiar os interessados de um projeto, devido a mudanças nos requisitos iniciais, erros encontrados, utilização de novas tecnologias e alterações na arquitetura [29]. Além disso, será adotado o conceito mais amplo em que a evolução não corresponde apenas às atividades pós-entrega do produto, mas também aquelas realizadas durante a fase inicial de desenvolvimento com o objetivo de facilitar a evolução e as atividades de aprendizado do sistema para implementar as mudanças requisitadas [30].

As próximas seções apresentam as leis que governam a evolução de software propostas por Lehman [23] (Seção 2.2.1), o processo de evolução (Seção 2.2.2), as características da evolução em linhas de produto de software (Seção 2.2.3), os conceitos básicos da disciplina de Gerência de Configuração (Seção 2.2.4) e as definições de variantes e versões de componentes de software (Seção 2.2.5).

2.2.1 As Leis de Evolução de Software

Lehman [23] [30], através de experimentos, estabeleceu o seguinte conjunto de leis que determinam a dinâmica da evolução de software:

1. ***Mudança Contínua:*** um sistema em produção deve manter-se em um contínuo processo de adaptação. Caso contrário, o sistema se tornará menos satisfatório.

2. ***Aumento de Complexidade:*** à medida que um sistema evolui, a sua complexidade aumenta. Com isso, atividades para preservar e simplificar a estrutura interna do sistema devem ser planejadas, evitando a erosão da arquitetura.

3. ***Auto-regulamentação:*** ao longo do tempo, atributos como tamanho, tempo entre liberações de versões e número de erros encontrados é praticamente invariante para cada versão do sistema. Como o processo é estatisticamente determinado, as atividades de criação e manutenção podem ser planejadas.

4. ***Conservação da Estabilidade Organizacional:*** ao longo do tempo, a taxa de desenvolvimento é aproximadamente constante e independente do número de recursos reservados a esta atividade.

5. ***Conservação da Familiaridade:*** durante o ciclo de vida de um sistema de software em evolução, o conteúdo de versões sucessivas é estatisticamente invariante.

6. ***Crescimento Contínuo:*** as funcionalidades disponibilizadas por um sistema devem continuar crescendo para manter a satisfação do cliente.

7. ***Redução da Qualidade:*** os sistemas precisam considerar as mudanças no ambiente operacional, evitando uma percepção de queda na sua qualidade.

8. ***Sistema de Opinião***[18]***:*** o processo de evolução consiste num ciclo de opiniões que devem sem consideradas para a melhoria do sistema.

O número de dados quantitativos para apoiar as leis de Lehman é pequeno, no entanto existem muitos dados experimentais [24]. Por isso, é importante levar em consideração essas leis na definição de um processo de apoio a evolução de software em uma organização.

2.2.2 O Processo de Evolução de Software

As *requisições de mudanças*[19] são responsáveis pelo direcionamento da evolução de um sistema. Um *processo de evolução* corresponde a um conjunto de atividades que são responsáveis pelo gerenciamento das mudanças solicitadas para um sistema. O processo de evolução varia de acordo com o tipo de software, o processo de desenvolvimento adotado pela organização e as pessoas envolvidas. Em algumas empresas, ele pode ser informal a ponto de as requisições serem solicitadas a partir de conversas entre os usuários do sistema e os desenvolvedores. Em outros casos, existe um processo formal, em que é necessário

[18] do inglês, *feedback*
[19] do inglês, *change request*

gerar uma documentação específica para cada fase do desenvolvimento [30] [24].

O processo de evolução, ilustrado na Figura 2.8, inclui as atividades de análise das mudanças solicitadas, planejamento das entregas[20], implementação e implantação do sistema atualizado no ambiente do cliente. O custo e o impacto das mudanças são avaliados de forma a se identificar o quanto do sistema como um todo será afetado e o custo estimado de se implementar a alteração, permitindo uma análise dos riscos envolvidos. Se as requisições são aceitas, uma nova entrega será planejada. Durante este planejamento, todas as mudanças solicitadas (adaptações, correções de erros, novos requisitos) são analisadas para decidir quais irão fazer parte da próxima entrega e aquelas que serão postergadas ou canceladas. As mudanças escolhidas são implementadas e validadas e uma nova versão do sistema é disponibilizada. Conforme ilustrado na Figura 2.9, o processo de identificação de mudanças e de evolução é cíclico e continua durante todo o ciclo de vida do sistema [30].

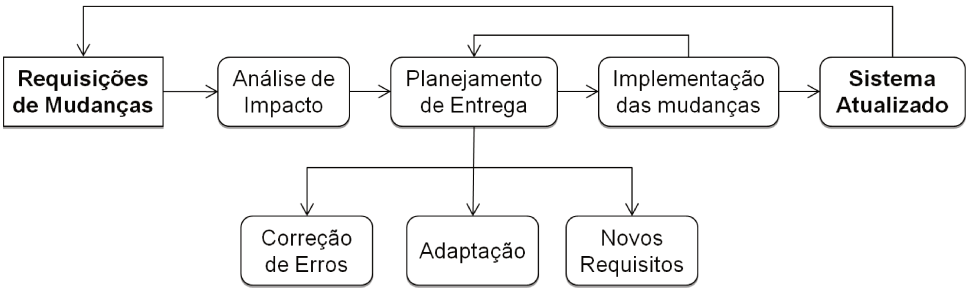

Figura 2.8: O processo de evolução de software.

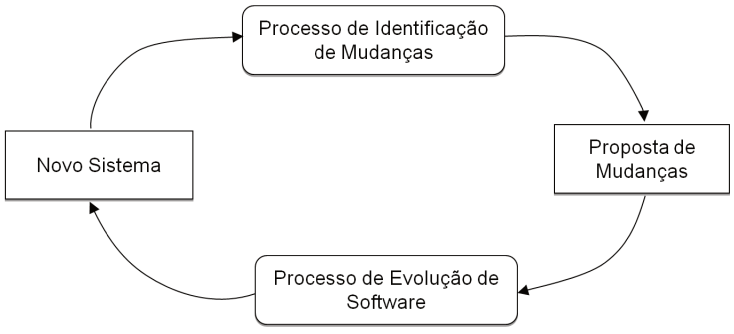

Figura 2.9: O processo de identificação de mudanças em software.

Durante a fase de implementação das mudanças, ilustrada na Figura 2.10, a especificação, o projeto e a implementação do sistema devem ser atualizados para refletir as

[20]do inglês, *release*

mudanças solicitadas. Entretanto, algumas requisições de mudança podem estar relacionadas a problemas do sistema que devem ser corrigidos o mais rápido possível, como quando um erro impede que o usuário execute o principal caso de uso do sistema. Neste caso, as mudanças precisam ser implementadas de forma mais rápida e o processo de análise formal e atualização de documentos não é realizado.

Figura 2.10: A etapa de implementação do processo de evolução de software.

2.2.3 Evolução em Linhas de Produto de Software

A evolução em uma linha de produto de software é mais complicada devido à possibilidade da evolução de um único ativo afetar muitos outros ativos e múltiplos produtos [24]. A Figura 2.11 ilustra os possíveis efeitos de uma mudança nos ativos de uma linha de produto. Por exemplo, a evolução da arquitetura provavelmente irá exigir alguma adaptação dos componentes existentes [32].

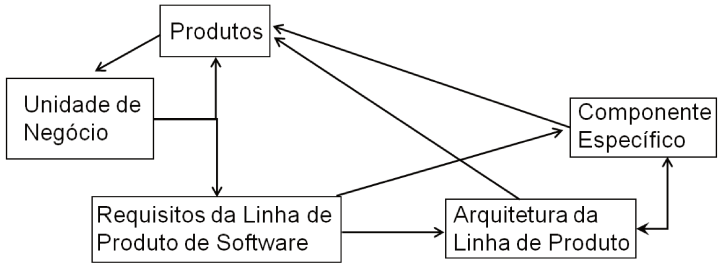

Figura 2.11: Propagação dos efeitos de uma mudança entre os ativos de uma LPS.

Em uma linha de produto de software, a evolução pode ocorrer por influências externas ou internas [24]. No caso das influências externas, pode-se citar:

- *competidores* que ao colocar um produto inovador no mercado, podem exigir que a linha de produto atual evolua de forma a manter a empresa competitiva.
- *clientes* que exigem que os produtos utilizem uma tecnologia mais recente.
- *fornecedores de componentes* que podem descontinuar ou evoluir um componente utilizado por uma linha de produto, para atender a exigências de mercado.

As influências internas ocorrem devido à interação existente entre os seguintes responsáveis pelas atividades essenciais de uma linha de produto de software:

- *desenvolvedores dos ativos centrais:* geram novas versões e variantes dos componentes, devido a requisições dos desenvolvedores de produtos e gerentes e de erros encontrados durante a realização de testes. Esta evolução impacta os desenvolvedores de produtos, pois eles precisam entender os novos processos, procedimentos e interfaces relacionados aos componentes.

- *desenvolvedores de produtos:* solicitam requisições de mudanças para os componentes que utilizam, para atender a exigências do mercado, implementar melhorias ou corrigir erros encontrados durante a execução de testes. Além disso, podem exigir que componentes específicos sejam integrados aos ativos centrais.

- *gerentes:* impactam os desenvolvedores dos ativos centrais através da exigência da utilização de novas tecnologias e mudanças no plano de negócios da linha de produto. Além disso, ao realizar mudanças de escopo, exigem que os desenvolvedores de produtos alterem os seus produtos para atender as novas expectativas.

A Figura 2.12 apresenta as áreas de conhecimento sugeridas pelo padrão de linha de produto *Evoluir Cada Ativo* (Seção 2.1.3), que possui como objetivo prover um suporte adequado à evolução dos ativos de uma linha de produto de software.

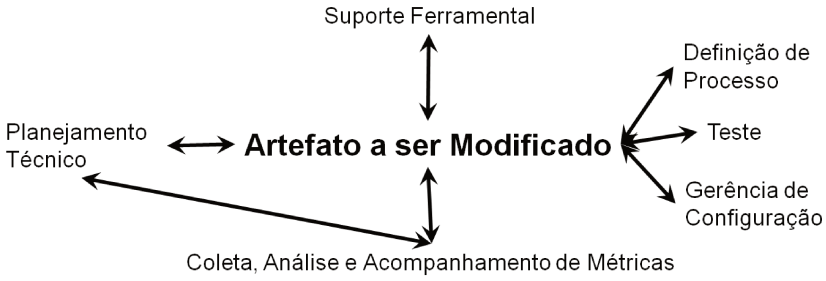

Figura 2.12: Áreas de conhecimento relacionadas a evolução de um ativo em uma LPS.

Nesta dissertação, o ativo a ser modificado é o componente afetado por uma requisição de mudança realizada por qualquer linha de produto que o utiliza. Conforme ilustrado na Figura 2.12, o domínio das seguintes áreas é necessário para que a evolução ocorra de maneira adequada: suporte ferramental, planejamento técnico, coleta e análise de métricas, teste, práticas de gerência de configuração e definição do processo para a utilização do componente após a evolução.

2.2.4 Gerência de Configuração de Software

A *Gerência de Configuração de Software* é a disciplina responsável por organizar e controlar o desenvolvimento de sistemas. Com isso, o controle da *evolução* de um componente é um tópico da gerência de configuração que tem por objetivo garantir que as mudanças nos artefatos do desenvolvimento de software sejam realizadas de uma maneira sistemática e rastreável, de modo que o sistema de software esteja sempre em um estado bem definido e consistente [11].

No contexto de gerência de configuração de software, duas dimensões são relevantes: *variação no espaço* e *variação no tempo*. Na engenharia de software tradicional, apenas a dimensão do tempo é considerada e o conjunto de atividades desenvolvidas para gerenciá-la é chamada de *gerenciamento de configuração*. Em linhas de produto, a gerência de configuração é multidimensional e é conhecida por *gerenciamento de variação*[21] [19] [22].

2.2.5 Versões e Variantes de Componentes de Software

Uma *versão de um componente* é uma instância específica no tempo, devido a uma *revisão* ou *mudança*. A variação no tempo é relativa ao gerenciamento da evolução e mudança de requisitos ao longo do tempo e é usualmente realizada através do *versionamento*. A forma como um identificador de versão é atribuído a um componente está definida no seu *modelo de versionamento*. Em geral, os identificadores de versão consistem de números no formato XX.YY.ZZ. O modelo de versionamento mais utilizado incrementa o valor de XX quando mudanças de grande impacto ocorrem, enquanto correções de erros e melhorias incrementam os demais números [31].

O termo *variante* é utilizado para diferenciar componentes que possuem características distintas do ponto de vista do usuário. Entretanto, nesta dissertação, essa definição será expandida para diferenciar também componentes que implementam o mesmo conjunto de características de formas diferentes. Com isso, não apenas o ponto de vista do usuário deve ser considerado, mas também o ponto de vista não-funcional. Uma `versão` seria apenas a representação de um mesmo componente ao longo do tempo. O termo variante está associado à dimensão espacial, onde variações de um componente são mantidas ao mesmo tempo e cada uma pode gerar novas versões ou variantes. A variação no espaço é relativa ao gerenciamento das diferenças entre o uso do software em múltiplos contextos em um dado momento, como nos vários produtos de uma linha de produto de software [19] [22].

[21] do inglês, *variation management*

Na Figura 2.13, o eixo x representa a dimensão do tempo e as versões de um componente, enquanto o eixo y representa as suas variantes. Neste exemplo, observa-se que cada variante evolui de forma independente das demais e que cada uma possui alguma propriedade que as difere entre si, representada pelas diferentes formas geométricas [7].

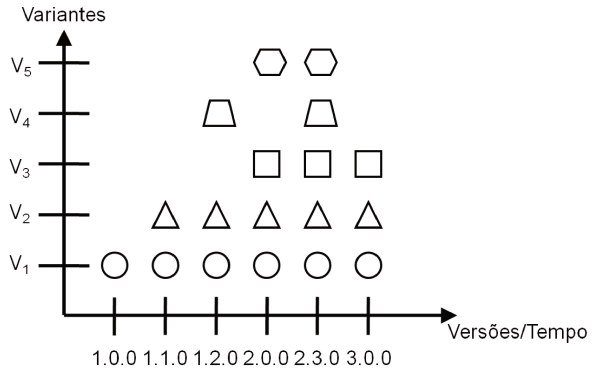

Figura 2.13: Versões e variantes de um mesmo componente.

Neste trabalho, um componente de software será representado por <*#variante,#versão*>, onde:

- *#variante*, corresponde ao identificador da variante de um componente. Este identificador será representado por <*nome do componente*>.*XXX*, onde:
  - <*nome do componente*> representa o nome do componente. Neste caso, apenas um acrônimo em letras maiúsculas deve ser utilizado. Por exemplo, se o nome do componente é `Mensagens`, pode-se utilizar `MSG`. Desta forma, pode-se identificar todas as variantes de um mesmo componente;
  - *XXX* identifica a variante do componente, sendo que a uma variante podem ser atribuídos valores entre `001` e `999`. Quando uma nova variante é criada, o próximo maior número disponível entre as variantes existentes deve ser utilizado.
- *#versão*, corresponde ao identificador de versão do componente, associada a variante especificada pelo identificador *#variante*. Este identificador será utilizado de acordo com o modelo de versionamento *XX.YY.ZZ*, em que o valor de *XX* é incrementado quando mudanças de grande impacto ocorrem, enquanto correções de erros e melhorias incrementam os demais números. Quando uma nova variante é criada, o identificador de versão inicial deve ser `01.00.00`;
- <, , e > são caracteres separadores.

A utilização do identificador proposto deixa mais clara a independência existente entre as variantes de um mesmo componente, pois cada uma pode continuar gerando novas versões sem se preocupar com a existência das demais variantes. A Figura 2.14 expande a representação de um componente apresentada na Seção 2.1.1 para incluir o seu identificador. Neste exemplo é utilizado o identificador `<X.001,04.YY.ZZ>`, onde o componente representado é o `Componente X` em sua variante `001`. Além disso, é possível perceber que a variante ilustrada está em sua versão `04.YY.ZZ`.

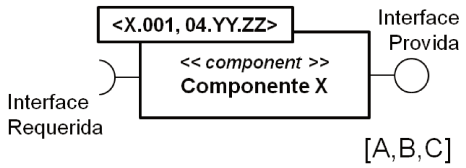

Figura 2.14: A representação de um componente de software com o seu identificador.

Algumas considerações sobre o formato acima:

- quando a versão de um componente não é importante para o contexto em que ele está sendo utilizado, pode-se fazer referência apenas ao identificador da variante. Uma outra forma é manter o formato proposto, mas inserir *XX.YY.ZZ* no identificador de versão. Desta forma, diz-se que a versão não é importante no contexto atual;

- não é possível identificar um componente apenas por seu número de versão, pois este identificador pode ser igual entre as variantes existentes.

- no caso de variantes temporárias, criadas para atender a algum objetivo específico de projeto e que são descontinuadas após um determinado período, o identificador da variante deve ser precedido por "_"e letras para identificar a nova variante temporária. Por exemplo, uma variante temporária da variante `MSG.001` seria `MSG.001_A`. Desta forma, torna-se possível identificar todas as variantes temporárias associadas a uma determinada variante do componente.

Os pontos de variação também devem ser versionados, pois podem evoluir com o tempo. Nesta dissertação, um ponto de variação será representado por *<PV_<nome do componente>.#versão>*, onde:

- *PV_* indica que se trata de um ponto de variação;

- *<nome do componente>* representa o nome do componente associado ao ponto de variação. Neste caso, apenas um acrônimo em letras maiúsculas deve ser utilizado.

Por exemplo, se o nome do componente é `Mensagens`, pode-se utilizar `MSG`. Este nome deve ser igual ao utilizado pelas variantes do componente;

- *#versão* utilizará o tradicional modelo de versionamento *XX.YY.ZZ*;
- <, , e > são caracteres separadores

A Figura 2.15 expande a representação da arquitetura de linha de produto apresentada na Seção 2.1.4 para incluir o identificador de versão no ponto de variação correspondente à escolha da variante do componente `Mensagem` a ser utilizada.

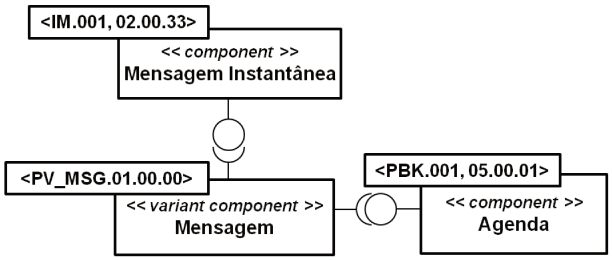

Figura 2.15: Versionamento de pontos de variação em uma ALP.

Capítulo 3

Evolução de Componentes Compartilhados por Múltiplas Linhas de Produto de Software

Este capítulo apresenta uma visão geral da família de padrões de evolução identificada no ambiente de desenvolvimento em que múltiplas linhas de produto de software são mantidas em paralelo. A Seção 3.1 apresenta o contexto que motivou o desenvolvimento deste trabalho. A Seção 3.2 apresenta os padrões de evolução identificados e o relacionamento entre eles e os padrões de linhas de produto de software existentes. A Seção 3.3 apresenta uma abordagem para auxiliar na escolha de uma variante de um componente a partir do conjunto de características desejado. Além disso, a Seção 3.4 apresenta um fluxograma para auxiliar os gerentes de linhas de produto e os gerentes responsáveis pelo desenvolvimento dos ativos centrais a decidir pelas abordagens de evolução identificadas. Por fim, na Seção 3.5, através de um exemplo ilustrativo, os padrões de evolução são apresentados de uma forma simples.

3.1 Motivação: O Ambiente de Desenvolvimento da Motorola

A motivação para este estudo foi o ambiente de desenvolvimento da empresa *Motorola Inc*, divisão de aparelhos móveis, onde múltiplas linhas de produto de software similares são mantidas para gerar o software embarcado dos celulares. Para compreender o porquê da abordagem de múltiplas linhas de produto de software e o compartilhamento de componentes entre elas, é preciso conhecer o processo de desenvolvimento e a organização interna da empresa.

O processo de desenvolvimento de celulares na Motorola pode ser dividido nas seguintes fases:

1. ***Definição dos requisitos funcionais e não-funcionais que caracterizam um grupo de celulares a ser desenvolvido.*** De forma geral, vários modelos de celulares são derivados de um mesmo conjunto de características. Estes modelos diferem em seu design externo e nas diferentes composições de características opcionais e alternativas disponíveis.

2. ***Escolha da arquitetura de linha de produto base para o desenvolvimento.*** A Motorola possui várias arquiteturas que podem ser utilizadas como base para o desenvolvimento de novos celulares. Por exemplo, existem arquiteturas que favorecem o desenvolvimento de celulares com enfoque em aplicações multimídia, voltados ao mercado *low-cost* ou que possuem telas sensíveis ao toque.

3. ***Definição da linha de produto que irá apoiar o desenvolvimento do software embarcado dos novos celulares***. De uma forma geral, uma linha de produto existente é utilizada como base para a criação de uma nova linha que irá apoiar o desenvolvimento de novos conjuntos de celulares. Esta abordagem favorece a manutenção de várias linhas de produto em paralelo e o alto grau de similaridade entre elas.

4. ***Implementação das novas características e estabilização da linha de produto.*** Esta fase corresponde ao desenvolvimento das novas características requisitadas na fase inicial e ao conjunto de validações que serão realizadas antes dos celulares serem entregues aos clientes.

A organização das equipes de desenvolvimento de software na empresa segue a estrutura sugerida na literatura de linha de produto. Na Figura 3.1, pode-se ver a divisão das equipes nas áreas de "Desenvolvimento de Produtos" e "Desenvolvimento de Ativos Centrais". Assim, as equipes responsáveis por um produto específico ou por um grupo de produtos solicitam as mudanças necessárias para as equipes responsáveis pelos componentes existentes na arquitetura utilizada.

A Figura 3.2 ilustra o paralelismo existente no desenvolvimento dos vários conjuntos de produtos e a necessidade das equipes dos componentes de apoiar vários produtos. Esses produtos podem ser baseados em arquiteturas diferentes, fazer parte de linhas de produto distintas e possuir características diversas e por vezes conflitantes. O exemplo da Figura 3.2 ilustra duas variantes do componente `Instant Messaging` (variantes `IM.001` e

IM.002) que são utilizadas em 3 arquiteturas de linha de produto distintas (ALP A, ALP B e ALP C). A equipe responsável pelo seu desenvolvimento deve ser capaz de manter todas essas variantes.

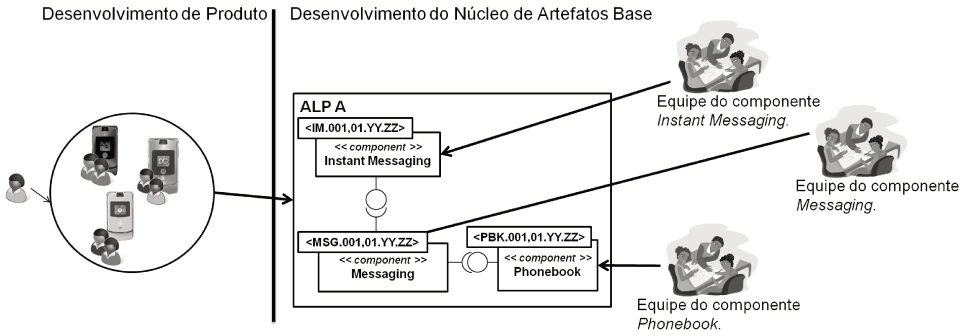

Figura 3.1: Estrutura organizacional da equipe de desenvolvimento da Motorola.

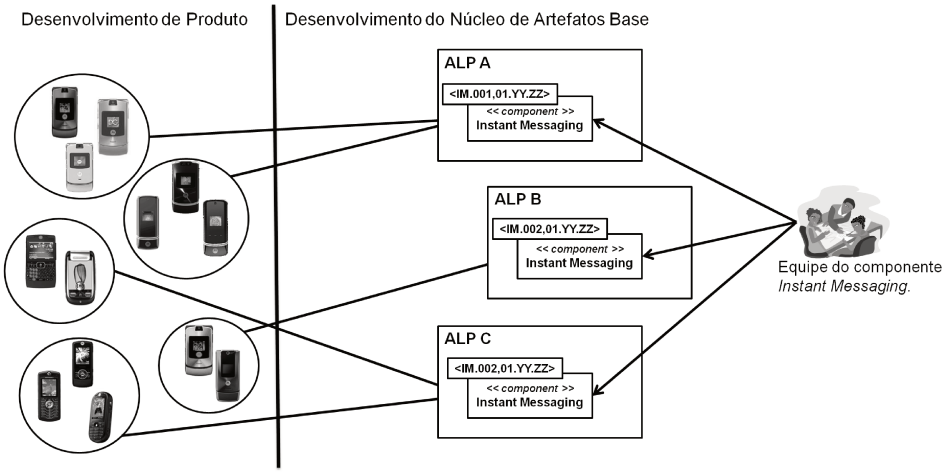

Figura 3.2: Ambiente de desenvolvimento de produtos da Motorola.

Para as equipes responsáveis pelo desenvolvimento dos componentes de software, o apoio a várias linhas de produto de software em paralelo representa um grande desafio. A evolução desses componentes exige uma análise criteriosa para evitar que as mudanças a serem realizadas impactem os produtos de uma determinada linha ou favoreçam uma linha de produtos em detrimento da outra. Além disso, o componente deve ser capaz de adaptar-se a diferentes plataformas de hardware e software e apoiar requisitos não-funcionais conflitantes. Por exemplo, um componente deve ser capaz de apoiar o desenvolvimento de celulares voltados ao mercado *high-end* e *low-end*, onde a capacidade de processamento e memória disponíveis é bem diferente. Neste caso, o componente não

pode se limitar a oferecer apenas os serviços apoiados por celulares *low-end*. Ele deve apoiar ambas as linhas, oferecendo serviços com a qualidade esperada para os dois casos. Com isso, o principal desafio dos responsáveis pelos componentes é decidir qual é a melhor forma de gerenciar a evolução para incluir mudanças requisitadas por uma linha de produto, com a preocupação de minimizar o impacto e, ao mesmo tempo, explorar ao máximo os recursos disponibilizados nas demais linhas.

Resumindo, este trabalho tem como enfoque analisar os cenários de evolução enfrentados pelas equipes de desenvolvimento de componentes da Motorola, no contexto do processo de desenvolvimento e da estrutura organizacional apresentados anteriormente. De uma forma geral, estes cenários podem ocorrer em qualquer organização que mantenha várias linhas de produto baseadas em componentes em paralelo, onde componentes são compartilhados entre as múltiplas linhas.

3.2 Visão Geral da Família de Padrões de Evolução

O relacionamento entre os padrões de evolução identificados por este trabalho e outros padrões de linhas de produto de software da literatura [10, 24] é apresentado na Figura 3.3. O padrão *Fábrica* determina as atividades necessárias para estabelecer e iniciar a produção em uma linha de produto de software [10].

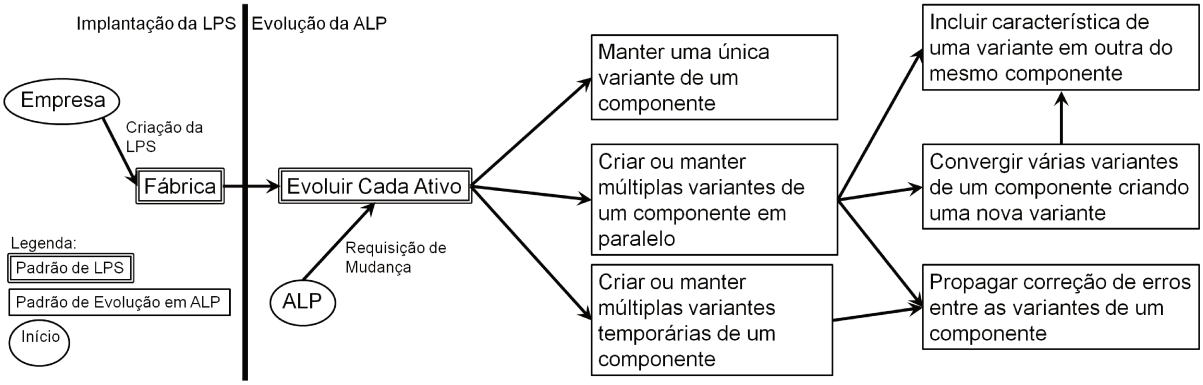

Figura 3.3: Família de padrões de evolução para componentes compartilhados por múltiplas LPSs.

O padrão *Evoluir Cada Ativo* [24] coordena as atividades necessárias para a implementação de mudanças em um ativo previamente construído. Neste trabalho, o ativo a ser modificado é o componente afetado por uma requisição de mudança realizada por

qualquer arquitetura de linha de produto que o utiliza. O padrão *Criar ou manter múltiplas variantes de um componente em paralelo* é uma derivação de *Evoluir Cada Ativo* e representa a opção pela abordagem de compartilhar componentes entre várias linhas de produto e a manter múltiplas variantes para implementar a variabilidade de software. O padrão *Manter uma única variante de um componente* corresponde à evolução do componente através da criação de uma nova versão ou revisão e ao uso de uma única variante simultaneamente por várias linhas de produto, ou seja, uma nova variante não é criada. A evolução através da criação de uma variante temporária, para atender a algum objetivo específico, é atendida pelo padrão *Criar ou manter múltiplas variantes temporárias de um componente*. De uma forma geral, este tipo de evolução ocorre quando os produtos fabricados por uma linha de produto de software estão em fase de teste de aceitação por clientes. Assim, qualquer correção de erro ou nova característica solicitada por um cliente, aumenta o risco de inserção de novos erros em um produto já considerado estável do ponto de vista de outro cliente.

Ainda na Figura 3.3, o padrão *Criar ou manter múltiplas variantes de um componente em paralelo* torna possível a ocorrência de outros padrões de evolução identificados por esta dissertação. A criação de uma nova variante a partir da unificação do conjunto de características de duas ou mais variantes é ilustrado pelo padrão *Convergir várias variantes de um componente criando uma nova variante*. A inclusão de características de uma variante do componente em outra variante é representada pelo padrão *Incluir característica de uma variante em outra do mesmo componente*. Nesta abordagem de evolução, uma linha de produto que utiliza uma determinada variante de um componente faz uma requisição de uma característica disponível em outra variante. Além disso, o padrão *Propagar correção de erros entre variantes de um componente* é uma consequência direta da manutenção de múltiplas variantes em paralelo. A principal motivação para a propagação da correção de erros é a necessidade de se manter estáveis as múltiplas variantes de um componente e evitar que erros conhecidos sejam reportados novamente em um ciclo de testes de outra variante.

3.3 Instanciando a Arquitetura da Linha de Produto de Software

As variantes de um determinado componente possuem diferentes vetores de características, com exceção do cenário onde duas variantes diferem apenas pela forma como implementam seus vetores. O conjunto de características de uma determinada arquitetura de linha de produto consiste das características disponibilizadas por seus componentes concretos e

pelas diversas variantes associadas aos pontos de variação de componentes existentes. Resumindo, o vetor de características de uma arquitetura de linha de produtos consiste dos conjuntos de:

- vetores de características de seus componentes concretos;
- todos os vetores de características das variantes de componentes associados a um ponto de variação na arquitetura.

A seleção da variante a ser utilizada é realizada com base nas características exigidas pelos produtos apoiados pela arquitetura de linha de produto. Para auxiliar nesta identificação, uma extensão do conceito de *Release Matrix* [34] será realizada. Em sua versão original, uma tabela é utilizada para mapear produtos a componentes, com o objetivo de identificar quais produtos seriam impactados por uma alteração em um determinado componente. Neste trabalho, a tabela será utilizada para mapear as variantes dos componentes às características existentes.

A Figura 3.4 ilustra uma arquitetura de linha de produto chamada `ALP A`, que possui os seguintes componentes: `Componente X`, `Componente Y` e `Componente Z`. Neste exemplo, o `Componente X` representa um ponto de variação na arquitetura. Desta forma, a variante do `Componente X` será escolhida após a definição do conjunto de características desejado pelos produtos a serem apoiados.

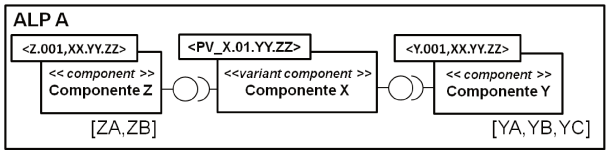

Figura 3.4: Arquitetura de Linha de Produto com pontos de variação a serem instanciados.

A Tabela 3.1 apresenta o relacionamento entre as variantes do `Componente X` e as características existentes. Neste exemplo, se a característica `I` e a `Característica F` devem estar disponíveis, a variante `<X.004,XX.YY.ZZ>` deve ser escolhida.

O conjunto de características da ALP A consiste do conjunto de características do `Componente Y`, `Componente Z` e de todas as características disponibilizadas pelas variantes do componente `Componente X`. Desta forma, o vetor de características da `ALP A` seria `[YA,YB,YC,ZA,ZB,XA,XB,XC,XD,XE,XF,XG,XH,XI]`.

Variantes	Características								
		XA	XB	XC	XD	XE	XF	XG	XH	XI
	<X.001,XX.YY.ZZ>	X	X	X	X	X				X
	<X.002,XX.YY.ZZ>	X	X	X	X	X	X	X	X	
	<X.003,XX.YY.ZZ>	X	X	X	X					X
	<X.004,XX.YY.ZZ>	X	X	X	X	X	X	X	X	X

Tabela 3.1: Relacionamento entre características e variantes do componente *Componente X*.

Após a escolha da variante a ser utilizada, é preciso garantir que todas as características desejadas estão habilitadas. De uma forma geral, uma característica pode estar desabilitada quando ela foi implementada como opcional ou alternativa dentro da própria linha de produto que a solicitou. Neste caso, os componentes utilizam de outros artefatos para permitir que uma característica possa ser desabilitada, como: diretivas de compilação e parâmetros de configuração. A resolução desses pontos de variação pode ocorrer em tempo de fabricação ou ser realizada pelos próprios clientes. O gerente da linha de produto é o responsável por garantir que todos os pontos de variação disponibilizados pelos componentes estejam resolvidos no produto final.

3.4 O Processo de Análise para a Evolução de um Componente

A Seção 3.1 apresentou o ambiente de desenvolvimento em que múltiplas linhas de produto de software são mantidas em paralelo e compartilham componentes entre si. Em seguida, a Seção 3.2 apresentou os padrões de evolução de componentes identificados neste contexto. Além disso, a Seção 3.3 apresentou a metodologia utilizada para a escolha de uma variante de um componente no momento da resolução dos pontos de variação existentes na arquitetura da linha de produto. Entretanto, ainda não foi definido um processo que auxilie a identificar qual o melhor padrão de evolução a ser adotado quando uma alteração nas características apoiadas pelos produtos de uma linha de produto é solicitada. Desta forma, esta seção tem como objetivo apresentar um fluxograma com as atividades necessárias para se escolher a melhor abordagem de evolução ao implementar uma requisição de mudança para uma arquitetura de linha de produto.

No contexto deste trabalho, para gerenciar as requisições de mudanças, os seguintes papéis precisam ser atribuídos entre as pessoas que fazem parte da equipe de desenvolvimento de software:

- *Gerentes de Linhas de Produto:* responsáveis por definir o conjunto de características que os produtos apoiados por uma determinada linha de produto irão possuir. A definição da arquitetura da linha de produto e a produção dos produtos planejados é de responsabilidade desse grupo. Além disso, os gerentes de linha de produto são responsáveis por resolver os pontos de variação de componentes presentes na arquitetura de linha de produto projetada.

- *Gerentes de Componentes:* um gerente de componente deve ser designado para cada componente presente na arquitetura de linha de produto. Desta forma, tornam-se responsáveis pela sua evolução. Os gerentes de componentes devem garantir que os componentes são desenvolvidos de modo a atender a múltiplas linhas de produto. A forma como a variabilidade interna é implementada é de responsabilidade do gerente de componente.

Conforme mencionado na Seção 2.2.3, uma requisição de mudança é influenciada por vários fatores externos e internos. Entretanto, o gerente de linha de produto é responsável por fazer uma seleção das requisições recebidas para os seus produtos e identificar aquelas que são válidas. Após essa seleção inicial, as mudanças relacionadas a melhorias, correções de erros e novas características devem seguir caminhos de análise distintos.

Com base no fluxograma da Figura 3.5, as requisições de mudanças consideradas inválidas são descartadas. As mudanças relacionadas a correções de erros são passadas diretamente aos gerentes dos componentes que implementam as características onde os problemas foram encontrados. Estes, por sua vez, podem:

- receber a requisição de mudança e corrigir o problema, seguindo o padrão *Propagar correção de erros entre variantes de um componente*;

- quando a requisição de mudança foi entregue de forma incorreta, devolvê-la para o gerente da linha de produto identificar outro componente que possa implementar a característica defeituosa;

- solicitar que outros gerentes de componentes auxiliem na implementação da mudança solicitada, quando a característica defeituosa é implementada parcialmente por vários componentes. Esta é mais uma aplicação do padrão *Propagar correção de erros entre variantes de um componente*.

Caso a mudança seja relacionada a melhorias em características existentes, o gerente de linha de produto pode decidir iniciar uma nova linha de produto para apoiar estas melhorias. Entretanto, de qualquer forma, a requisição de mudança será passada ao gerente de componente. Neste caso, ele irá decidir qual a melhor abordagem que deve ser seguida

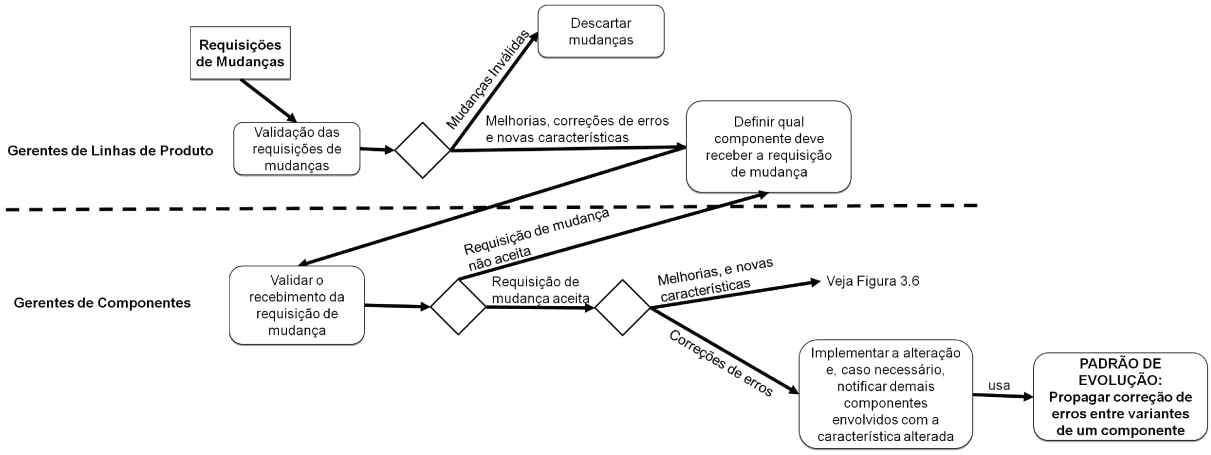

Figura 3.5: Processo de seleção de requisições de mudanças a serem implementadas e o fluxo de atividades relacionado a correção de erros.

entre os padrões *Criar ou manter múltiplas variantes de um componente em paralelo* e *Manter uma única variante de um componente*. Caso a característica já exista em alguma variante do componente utilizada por outras arquiteturas de linha de produto, os padrões *Incluir característica de uma variante em outra do mesmo componente* ou *Convergir várias variantes de um componente criando uma nova variante* podem ser utilizados. O fluxograma da Figura 3.6 ilustra este fluxo de atividades, que também é válido para a implementação de novas características.

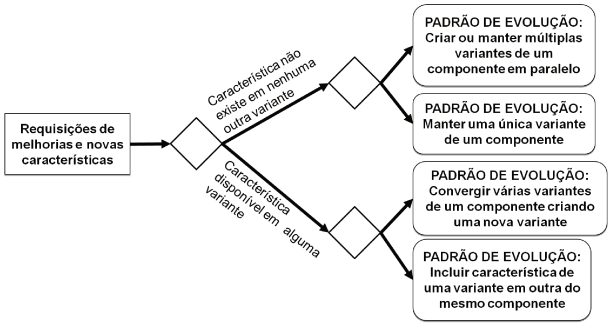

Figura 3.6: Fluxo de atividades relacionado a implementação de melhorias a características existentes e de novas características.

Além do fluxo apresentado na Figura 3.6, pode-se ainda definir um fluxo alternativo onde, apesar da característica sendo requisitada existir em alguma variante, o padrão *Criar ou manter múltiplas variantes de um componente em paralelo* é aplicado para criar uma nova variante e o padrão *Incluir característica de uma variante em outra do mesmo*

componente é utilizado logo em seguida para trazer a característica existente a nova variante. Este cenário pode ocorrer quando se deseja minimizar o risco de alterar a variante que estava sendo usada pela arquitetura da linha de produto que recebeu a requisição de nova característica, pois ela é usada por outras arquiteturas de linhas de produto.

O gerente de linha de produto também pode solicitar aos gerentes de componentes que alguma requisição de mudança seja realizada em variantes temporárias de componentes. Este tipo de requisição ocorre quando uma arquitetura de linha de produto não pode suportar o risco de receber mudanças que podem prejudicar um produto já considerado estável. Esta atividade corresponde ao padrão *Criar ou manter múltiplas variantes temporárias de um componente*.

Conforme ilustrado nos fluxogramas anteriores, a evolução ocorre, principalmente, devido a interação entre os gerentes de linha de produto e gerentes de componente. Quando uma nova linha de produto de software é criada, o gerente da linha de produto identifica o conjunto de características desejado e define a arquitetura da linha de produto, tendo como base os componentes já mantidos pela empresa. Para os componentes existentes, um subconjunto de características relacionadas é enviado ao gerente do componente. Nesta etapa, o gerente de componente deve analisar se existe alguma variante do seu componente que implementa as características desejadas. Caso exista, ela deve ser oferecida ao gerente da linha de produto. Neste cenário, a mesma variante pode estar sendo usada por mais de uma linha de produto. Caso o conjunto de características desejado não esteja implementado por algum componente, o gerente de componente deve decidir como proceder com a evolução do componente. A família de padrões de evolução identificada por este trabalho torna-se útil nesta situação. Os gerentes de linha de produto também podem solicitar a criação de variantes temporárias para os gerentes de produto, que serão responsáveis pelo seu gerenciamento.

3.5 Exemplo Ilustrativo: A LPS da Empresa Câmera & Cia

Os padrões de evolução para componentes da Figura 3.3 serão aplicados a um exemplo ilustrativo da empresa fictícia Câmera&Cia. Esta empresa desenvolve câmeras fotográficas digitais semi-profissionais e pretende ampliar o seu campo de atuação através da comercialização de câmeras digitais voltadas ao consumidor profissional.

3.5.1 Fábrica e Evoluir Cada Ativo

Os responsáveis pelo desenvolvimento de produtos da Câmera&Cia já utilizam a abordagem de linha de produto de software para as suas câmeras semi-profissionais. Para isso, as práticas sugeridas pelo padrão *Fábrica* foram seguidas. A Figura 3.7 apresenta a `ALP Câmera Semi-Profissional`, que é composta pelos componentes `Controlador de Lentes`, `Gerenciador de Imagens` e `Reconhecimento de Imagem` pertencentes aos ativos centrais.

Figura 3.7: *ALP Câmera Semi-Profissional* e seus componentes em UML.

Para esse exemplo, considere que devido ao sucesso obtido com a abordagem de linhas de produto de software, o desenvolvimento dos produtos voltados ao consumidor profissional também ocorrerá seguindo estes conceitos. Entretanto, uma nova linha de produtos será criada, pois:

- o conjunto de características associadas aos produtos profissionais é maior, exigindo a criação de novos componentes e alterações na arquitetura utilizada pela linha de produto existente.
- o hardware utilizado pelos produtos profissionais permite um maior poder de processamento e os componentes existentes devem ser customizados para aproveitar ao máximo os recursos disponibilizados.
- o grande número de alterações exigidas pelos produtos profissionais implica em alto risco de inserção de erros nos produtos voltados ao mercado semi-profissional, que constituem a principal fonte de renda da empresa.
- a linha de produto de software atual é estável, ou seja, os produtos produzidos apresentam uma baixa taxa de erros durante o seu ciclo de vida.

Baseado nas constatações acima, uma nova linha de produto para câmeras profissionais foi criada. Apesar da nova linha de produto implementar novas características, a sua semelhança com a linha de produto de câmeras semi-profissionais é muito grande e vários ativos podem ser compartilhados. Para implementar as novas características, as diretrizes apresentadas pelo padrão *Evoluir Cada Ativo* são utilizadas.

3.5.2 Criar ou manter múltiplas variantes de um componente em paralelo

No caso dos componentes afetados pela decisão de criar duas ALPs, a evolução será guiada pelo padrão *Criar ou manter múltiplas variantes de um componente em paralelo* da Figura 3.3. Assim, a variante `<RI.002,XX.YY.ZZ>` será criada para atender as novas características, de forma a implementá-las com um impacto baixo para a linha de produto de câmeras semi-profissionais. A Figura 3.8 ilustra a nova `ALP Câmera Profissional` e o componente comum `Reconhecimento de Imagem`. Neste caso, as variantes `<RI.001,XX.YY.ZZ>` e `<RI.002,XX.YY.ZZ>` do componente `Reconhecimento de Imagem` presentes no núcleo comum das linhas de produto de software serão mantidas em paralelo. O componente `Controlador de Lentes` continuará sendo utilizado por ambas as linhas em sua variante original, correspondendo ao padrão *Manter uma única variante de um componente* da Figura 3.3. O componente `Gerenciador de Imagens` continua sendo utilizado apenas pela `ALP Câmera Semi-Profissional`. Além disso, o componente `Gerenciador de Mídia` é criado e utilizado pela `ALP Câmera Profissional`.

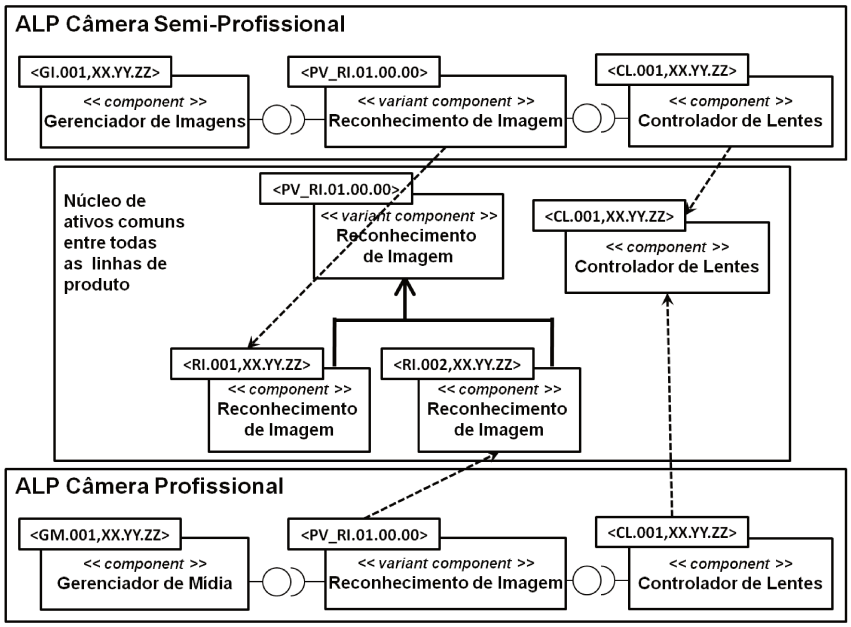

Figura 3.8: Criação da *ALP Câmera Profissional*.

Considere ainda que a empresa Câmera&Cia decida investir em câmeras para celulares. Como esta é uma área totalmente nova, com grandes restrições de características e dependência de hardware, uma nova linha de produto de câmeras celulares será criada. Além disso, devido ao conjunto de características requeridas ser semelhante ao disponibilizado

pela linha de produto de câmeras semi-profissionais, as alterações necessárias serão realizadas tendo como base os ativos desta linha de produto. A Figura 3.9 ilustra as três ALPs da empresa. Uma nova variante do componente `Reconhecimento de Imagem` é gerada e ele passa a ser utilizado pelas três arquiteturas. O componente `Controlador de Lentes` continua sendo utilizado por todas as linhas na forma da sua variante original e uma nova variante do componente `Gerenciador de Imagens` é criada. Com isso, ele também passa a ser compartilhado pela `ALP Câmera Celular` e `ALP Câmera Semi-Profissional` através da criação de múltiplas variantes.

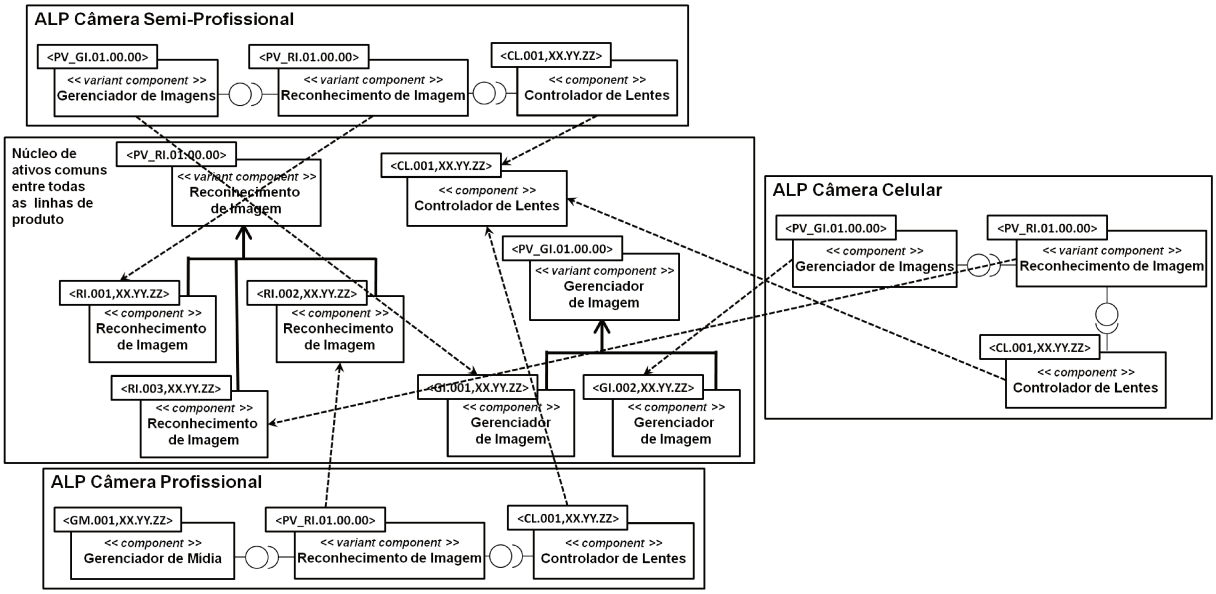

Figura 3.9: Criação da *ALP Câmera Celular*.

3.5.3 Criar ou manter múltiplas variantes temporárias de um componente

Após a linha de produto de câmeras celulares ser colocada em operação, os produtos fabricados começaram a ser entregues para clientes de diversas regiões para que fossem realizados testes de aceitação. Como cada cliente possui uma configuração de características específica, três variantes temporárias (`<RI.003_A,XX.YY.ZZ>`, `<RI.003_B,XX.YY.ZZ>` e `<RI.003_C,XX.YY.ZZ>`) foram criadas a partir da variante `<RI.003,XX.YY.ZZ>` do componente `Reconhecimento de Imagem`, para os clientes A, B e C da linha de produto, como ilustrado na Figura 3.10. Este padrão é utilizado para evitar que correções de erros e requisições de última hora solicitadas por um cliente impactem os produtos dos demais. Por isso, as variantes temporárias são modificadas somente depois de uma análise de

risco cuidadosa. Além disso, elas serão descontinuadas após a aprovação do produto pelo cliente.

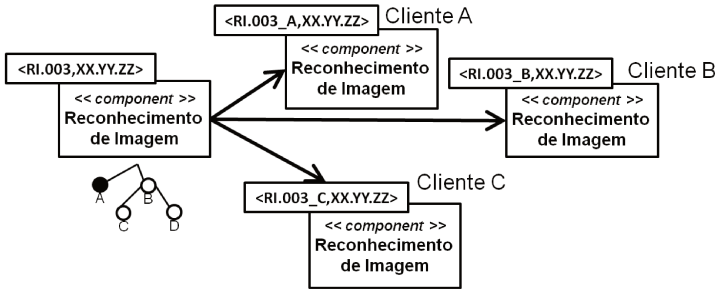

Figura 3.10: Variantes temporárias do componente *Reconhecimento de Imagem*.

3.5.4 Propagar correção de erros entre variantes de um componente

Durante os testes dos produtos fabricados pela linha de produto para câmeras semi-profissionais, foi descoberto um erro de implementação da característica de auto-ajuste disponibilizada pelo componente `Reconhecimento de Imagem`, representada pela característica `C` ilustrada na Figura 3.11. Assim, deve-se analisar se a correção aplicada a variante `<RI.001,XX.YY.ZZ>` do componente `Reconhecimento de Imagem` deve ser aplicada nas variantes `<RI.002,XX.YY.ZZ>`, `<RI.003,XX.YY.ZZ>`, `<RI.003_A,XX.YY.ZZ>`, `<RI.003_B,XX.YY.ZZ>` e `<RI.003_C,XX.YY.ZZ>`, pois todas possuem a característica `C` implementada. Esta situação ocorre com frequência em um ambiente com múltiplas variantes de um componente mantidas em paralelo.

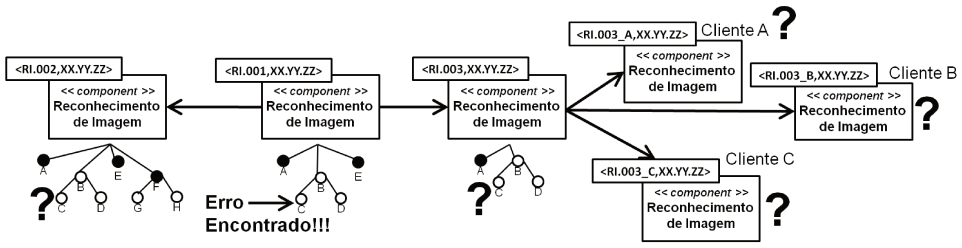

Figura 3.11: Impacto de um erro encontrado na variante *<RI.001,XX.YY.ZZ>* nas demais variantes do componente *Reconhecimento de Imagem*.

3.5.5 Incluir característica de uma variante em outra do mesmo componente

Com o objetivo de sempre oferecer novidades aos consumidores de suas câmeras semi-profissionais, a Câmera&Cia adicionou a característica de captura automática de fotos quando uma pessoa está sorrindo, representada pela característica `I` na Figura 3.12. Esta característica é implementada pela variante `<RI.001,XX.YY.ZZ>` do componente `Reconhecimento de Imagem`, gerando apenas uma nova versão ou revisão desta variante. Entretanto, devido ao sucesso desta nova característica, os clientes da linha de produto de câmeras celulares requisitaram a mesma característica em seus produtos. Como a variante do componente `Reconhecimento de Imagem` utilizada pela `ALP Câmera Celular` é a `<RI.003,XX.YY.ZZ>`, esta característica não está implementada e terá que ser incluída a partir da variante `<RI.001,XX.YY.ZZ>`. Este cenário corresponde ao padrão *Incluir característica de uma variante em outra do mesmo componente* da Figura 3.3. A simples substituição das variantes não pode ser realizada, pois o conjunto de características de ambas pode ter se modificado desde o momento da criação da variante `<RI.003,XX.YY.ZZ>`. A Figura 3.12 ilustra a inclusão da característica `I` da variante `<RI.001,XX.YY.ZZ>` do componente `Reconhecimento de Imagem` para a variante `<RI.003,XX.YY.ZZ>`. Neste padrão, uma nova variante não é criada, mas sim uma nova versão ou revisão da variante que está recebendo a nova característica. Por exemplo, a variante poderia sofrer uma alteração de `<RI.003,01.02.23>` para `<RI.003,01.03.00>`, ou seja, a inclusão da nova característica altera apenas o identificador de versão da variante `<RI.003>`.

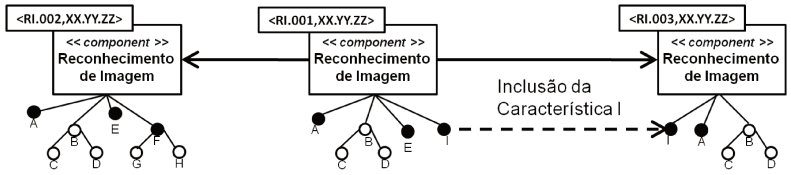

Figura 3.12: Inclusão da *Característica I* da variante *01.YY.ZZ* para a *03.YY.ZZ*.

3.5.6 Convergir várias variantes de um componente criando uma nova variante

O desenvolvimento de um novo conjunto de câmeras digitais está sendo analisado pela Câmera&Cia. Eles constituem um modelo intermediário entre o semi-profissional e o profissional já comercializados. Como o mercado para estes produtos é incerto e eles não substituirão os produtos atuais, uma nova linha de produtos será criada. Além disso, para evitar o impacto nas linhas de produtos em operação, que são responsáveis pelo fa-

turamento da empresa, novas variantes dos componentes existentes serão criadas. Como o conjunto de características desejado já foi implementado por componentes utilizados pelas `ALP Câmera Semi-Profissional` e `ALP Câmera Profissional`, uma nova variante será criada a partir das variantes utilizadas por cada uma dessas arquiteturas. Desta forma, o padrão *Convergir várias variantes de um componente criando uma nova variante* da Figura 3.3 será aplicado. Neste caso, duas ou mais variantes de um mesmo componente servirão como base para a criação de uma nova variante, sendo que o conjunto de características desta nova variante será constituído pelo superconjunto das características das variantes de origem. As variantes de origem podem ser substituídas ou mantidas em paralelo com a nova variante criada. Caso sejam substituídas, a aplicação sucessiva deste padrão de evolução pode fazer com que exista uma única variante do componente sendo compartilhada entre as diversas ALPs. Na Figura 3.13, a variante `<RI.004,XX.YY.ZZ>` foi criada tendo como base as variantes `<RI.001,XX.YY.ZZ>` e `<RI.002,XX.YY.ZZ>`, possuindo um superconjunto de suas características (a característica `F` da variante `<RI.002,XX.YY.ZZ>` e a característica `I` da variante `<RI.001,XX.YY.ZZ>`). Neste exemplo, todas as três variantes continuarão a evoluir em paralelo.

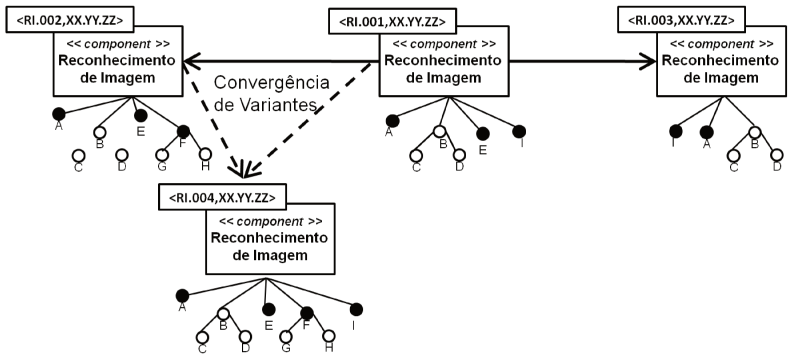

Figura 3.13: Convergência variantes do componente *Reconhecimento de Imagem*.

3.5.7 Resolvendo os pontos de variação na arquitetura das linhas de produto

No exemplo da empresa Câmera&Cia, o componente `Reconhecimento de Imagem` é um ponto de variação na `ALP Câmera Semi-Profissional`, `ALP Câmera Profissional`, `ALP Câmera Celular` e `ALP Câmera Profissional Intermediária`. A seleção da variante a ser utilizada é realizada com base nas características exigidas pelos produtos desenvolvidos em cada uma das linhas.

A Tabela 3.2 apresenta a relação de qual variante do componente `Reconhecimento de Imagem` disponibiliza determinada característica. Por exemplo, se uma arquitetura de linha de produto deseja o suporte a característica `E` e `G`, as variantes `<RI.002,XX.YY.ZZ>` ou `<RI.004,XX.YY.ZZ>` podem ser escolhidas.

Variantes	Características								
		A	B	C	D	E	F	G	H	I
	<RI.001,XX.YY.ZZ>	X	X	X	X	X				X
	<RI.002,XX.YY.ZZ>	X	X	X	X	X	X	X	X	
	<RI.003,XX.YY.ZZ>	X	X	X	X					X
	<RI.004,XX.YY.ZZ>	X	X	X	X	X	X	X	X	X

Tabela 3.2: Características X Variantes do componente *Reconhecimento de Imagem*.

Capítulo 4

A Família de Padrões de Evolução

Este capítulo apresenta em detalhes a família de padrões de evolução apresentada de forma ilustrativa no Capítulo 3. A Seção 4.1 apresenta o formato adotado para descrever os padrões de evolução. Os padrões encontrados são descritos em maiores detalhes e seguindo o formato proposto na Seção 4.2. Por fim, a Seção 4.3 ilustra, com exemplos reais da empresa Motorola Inc, o uso dos padrões de evolução identificados.

4.1 Representação dos Padrões de Evolução

O modelo abaixo, baseado no utilizado por `Gamma et al.` [16], é utilizado para descrever os padrões de evolução apresentados nesta dissertação. O seu principal objetivo é documentar de forma clara as características, conseqüências, alternativas e limitações de cada padrão, permitindo uma análise comparativa entre eles.

- **Nome do padrão de evolução**
  Descreve a essência do padrão de evolução de uma forma sucinta.
- **Descrição**
  Apresenta o padrão de evolução através de um exemplo e da descrição de suas características.
- **Motivação**
  Define o contexto técnico ou organizacional que pode fazer com que o responsável por um projeto decida por este padrão de evolução.
- **Conseqüências**
  Apresenta os *trade-offs* e resultados esperados ao se decidir por esse padrão de evolução, além do impacto da abordagem de evolução nos diversos artefatos do componente.

- **Relacionamento com outros padrões de evolução**
  Descreve a forma como o padrão atual se relaciona com os demais padrões de evolução identificados, apresentando as principais diferenças entre eles e formas de utilização em conjunto.

4.2 Família de Padrões de Evolução

As seguintes subseções descrevem os padrões de evolução identificados no contexto deste trabalho, seguindo o formato proposto na Seção 4.1.

4.2.1 Criar ou manter múltiplas variantes de um componente em paralelo

Descrição

Neste padrão de evolução, as requisições de mudança são implementadas através da criação de uma nova variante do componente existente. Desta forma, a variabilidade é implementada com o uso de variantes. Um único componente pode dar origem a diversas outras variantes que serão mantidas em paralelo a original e com ciclo de vida próprio, podendo sofrer outras formas de evolução. Neste cenário, a similaridade do conjunto de características e da implementação entre as diferentes variantes é muito grande. As novas variantes podem:

- *ser idênticas à anterior com relação ao número de características disponíveis, mas diferir na forma de implementação.* Esta variação pode ocorrer quando é necessário adaptar um componente a diferentes atributos de qualidade, sistemas operacionais ou realizar refatorações complexas.

- *possuir menos características que a variante anterior.* Por exemplo, na adaptação de um componente para uma configuração de hardware mais simples, pode ser necessário remover efetivamente o código de características não utilizadas.

- *possuir mais características que a variante anterior.* Por exemplo, novas características podem ser necessárias pois o componente será utilizado em uma linha de produtos de alta tecnologia.

Na Figura 4.1, pode-se ver o cenário tradicional, onde o componente `Componente X` é utilizado pela arquitetura de linha de produto `ALP A` em sua variante `<X.001,XX.YY.ZZ>`. A Figura 4.2, ilustra o cenário onde o componente `Componente X` tem duas variantes (`<X.001,XX.YY.ZZ>` e `<X.002,XX.YY.ZZ>`), sendo que cada uma é utilizada por uma

arquitetura de linha de produto diferente (`ALP A` e `ALP B`) para implementar diferentes conjuntos de características. Como as duas variantes dos componentes são necessárias, elas serão mantidas e poderão sofrer novas evoluções em paralelo.

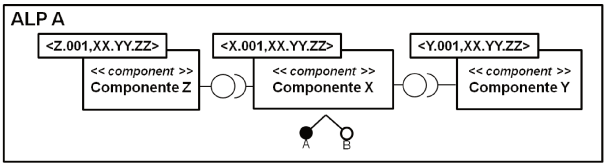

Figura 4.1: Arquitetura de Linha de Produto tradicional.

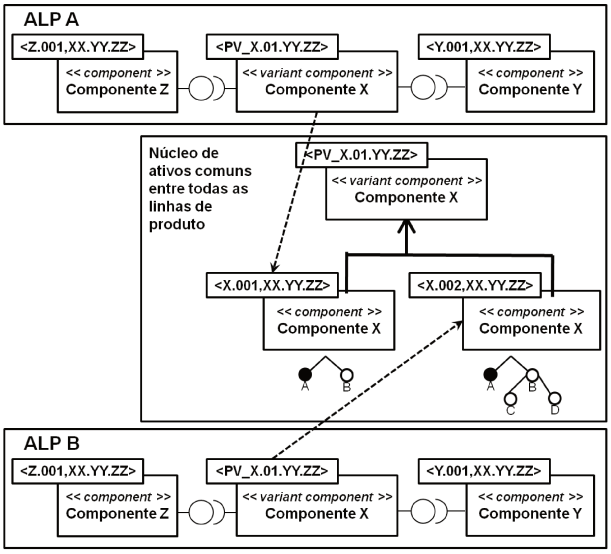

Figura 4.2: Múltiplas arquiteturas de linhas de produtos compartilhando um componente.

Motivação

`Bosch et al.` [9] [8], baseados em estudos de casos, descreveram algumas situações que podem levar à manutenção de múltiplas variantes (implementações) de um mesmo componente em paralelo:

- *Requisitos de qualidade conflitantes:* os diferentes produtos de uma linha de produto, mesmo quando requerendo a mesma característica, podem apresentar requisitos de qualidade conflitantes, como: desempenho, tamanho de código ou uso de memória. Em alguns casos, esses requisitos podem ser tão importantes que não é possível desenvolver um único componente capaz de satisfazer a ambos.

- *Variabilidade implementada através de variantes:* alguns tipos de variabilidade são difíceis de serem implementadas através de configurações ou diretivas de compilação, pois impactam o componente como um todo. Um exemplo de tal situação é a utilização de um mesmo componente em diferentes sistemas operacionais.

- *Produtos high-end versus low-end:* as variantes de baixo custo de um produto necessitam de uma quantidade menor de características. Desta forma, um componente com um menor conjunto de características pode ser necessário devido a restrições impostas pelo hardware.

Além dessas situações, pode-se ilustrar o caso em que um componente é considerado *estável*, ou seja, apresenta um baixo número de erros ao longo do período de desenvolvimento dos produtos atuais. Neste contexto, a adição ou alteração de características pode representar um alto risco de inserção de erros no componente. Com isso, a criação de uma nova variante para ser usada pelas linhas de produtos que requisitaram as mudanças passa a ser uma alternativa válida.

Conseqüências

O suporte a múltiplas variantes de um componente, dentro do contexto de linhas de produto, exige um gerenciamento de configuração mais complexo. Os seguintes artefatos precisam ser alterados para refletir a existência de mais de uma variante de um mesmo componente na linha de produto:

- *Requisitos:* a informação de que uma característica está incluída ou não em determinada variante de um componente deve estar presente em todos os documentos utilizados na especificação de requisitos.

- *Arquitetura Baseada em Componentes:* as múltiplas variantes de um componente podem interagir com outros elementos arquiteturais de diferentes formas. Assim, as interfaces e conectores exigidos por cada variante podem não ser os mesmos. Desta forma, a documentação da arquitetura, bem como as decisões arquiteturais, devem ser realizadas de forma a garantir o suporte adequado às várias variantes de um componente.

- *Projeto "Detalhado":* a implementação de uma determinada característica pode ser feita de forma diferente nas variantes de um componente. O documento de projeto deve ser capaz de apoiar diferentes visões para cada variante do componente, permitindo uma análise adequada da forma como a característica é implementada.

Relacionamento com outros padrões de evolução

A partir deste cenário de evolução, os padrões *Propagar correção de erros entre variantes de um componente* (Seção 4.2.3), *Convergir várias variantes de um componente criando uma nova variante* (Seção 4.2.4) e *Incluir característica de uma variante em outra do mesmo componente* (Seção 4.2.5) podem ocorrer.

4.2.2 Criar ou manter múltiplas variantes temporárias de um componente

Descrição

Este cenário ocorre da necessidade de se criar variantes temporárias para determinados componentes, com um objetivo específico. Apesar de o termo temporário estar sendo usado, uma variante pode durar semanas ou meses. Entretanto, as variantes temporárias irão ser descontinuadas quando o seu objetivo inicial for atingido. De uma forma geral, este tipo de evolução ocorre quando os produtos fabricados por uma linha de produto de software estão em fase de teste de aceitação por clientes. Dessa forma, qualquer correção de erro ou nova característica solicitada por um cliente, aumenta o risco de inserção de erros em um produto já considerado estável do ponto de vista de outro cliente. Assim, torna-se necessário uma abordagem de evolução que permita que apenas as mudanças necessárias para um determinado cliente sejam integradas.

As principais diferenças entre essa abordagem de evolução e a simples cópia de um componente pelo time de produto para realizar as alterações necessárias são:

- as alterações realizadas em uma variante temporária devem ser propagadas para a linha de evolução principal do componente. Com isso, a variante oficial do componente possuirá todas as alterações realizadas pelas várias variantes temporárias existentes.
- o time responsável pelo componente continua gerenciando a evolução das variantes temporárias e garantindo a consistência com a linha de evolução principal.

A Figura 4.3 apresenta um cenário hipotético de manutenção de múltiplas variantes temporárias de um componente. Nesta Figura, pode-se notar o fato das variantes temporárias sempre integrarem as suas alterações na variante oficial antes de serem descontinuadas. Entretanto, a atualização na linha principal não precisa ocorrer simultaneamente às alterações na ramificação temporária, como é mostrado na quarta ramificação[1].

[1] do inglês, *branch*

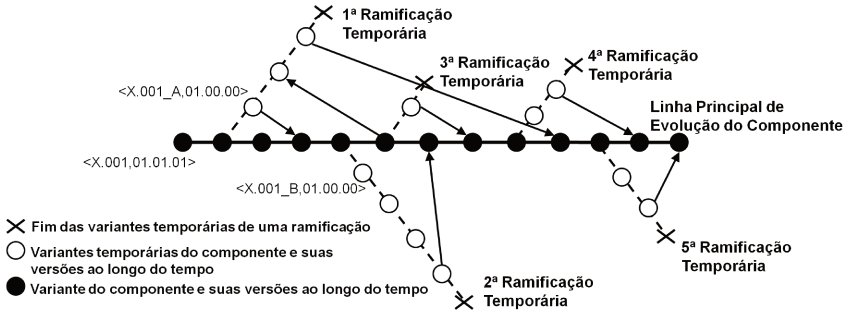

Figura 4.3: Múltiplas variantes temporárias de um componente.

Ainda na Figura 4.3, no caso da primeira ramificação temporária, pode-se notar uma atualização de variante com alterações provenientes da linha principal. Este tipo de evolução pode ocorrer devido a identificação de alguma mudança na linha principal considerada crítica para o cliente da ramificação temporária.

Motivação

Ommering[37], baseado em pesquisas realizadas em linhas de produto da *Philips Eletronics*, apresenta como principal motivação para este tipo de evolução o fato de não ser possível assumir o risco de incluir em um produto prestes a ser lançado características que serão necessárias para um produto futuro. Neste caso, apenas correções de erros solicitadas pelo cliente deveriam ser integradas.

Conseqüências

O suporte a múltiplas variantes temporárias de um componente, dentro do contexto de linhas de produto, exige um gerenciamento de configuração tão complexo quanto o mencionado no cenário *Criar ou manter múltiplas variantes de um componente em paralelo* (Seção 4.2.1). Apesar das variantes serem temporárias, é preciso garantir que as mudanças realizadas nelas serão propagadas para a linha principal. Com isso, apenas a documentação associada a linha principal precisa ser atualizada, pois as demais variantes serão descontinuadas. Como os novos produtos são derivados da linha principal, a sua consistência deve ser mantida para evitar retrabalhos.

Relacionamento com outros padrões de evolução

Este cenário é uma variação da abordagem *Criar ou manter múltiplas variantes de um componente em paralelo* (Seção 4.2.1). As diferenças estão relacionadas ao fato das variantes temporárias sempre serem descartadas e estarem ligadas à variante principal. En-

quanto no cenário *Criar ou manter múltiplas variantes de um componente em paralelo*, as novas variantes podem evoluir de forma independente e até mesmo gerar outras.

4.2.3 Propagar correção de erros entre variantes de um componente

Descrição

Quando uma correção de erro é realizada em uma determinada variante de um componente, deve-se analisar a necessidade de propagar essa correção para as demais variantes existentes. Para identificar as variantes que precisam receber a propagação de uma correção de erro, os seguintes itens podem ser analisados:

- *Documentos de evolução do componente*. Neste caso, pode ser utilizado qualquer documento que mostre as variantes correntes de um componente e suas origens. Com isso, é possível identificar possíveis candidatas a receber a propagação.
- *Documentos de requisitos*. Estes documentos são utilizados para verificar se a característica onde o erro foi encontrado está presente em outras variantes.
- *Documentos de projeto*. Permitem verificar se a implementação da característica onde o erro foi encontrado também é utilizada em outras variantes. Caso seja, a probabilidade da propagação ser necessária é maior.

A análise dos documentos acima permite filtrar o número de opções disponíveis e automatizar alguns passos da análise. Entretanto, a verificação manual (execução de testes e inspeção visual do código) da reprodutibilidade de um erro em uma determinada variante ainda será necessária na maioria dos casos. Além disso, nem todos os documentos citados anteriormente podem estar disponíveis.

Uma característica importante deste cenário de evolução é que o mesmo erro pode ser reproduzido em diferentes variantes, mas exigir correções diferentes para cada caso. Nesta situação, a propagação da correção deixa de ser trivial e uma nova análise para resolver o problema pode ser necessária. Isto pode ocorrer devido a diferenças na implementação de uma mesma característica entre as diferentes variantes de um componente.

A Figura 4.4 apresenta um diagrama parcial da evolução do componente `Componente X`. Na figura, as setas indicam que uma variante serviu de base para a criação da outra. Quando um erro é encontrado na característica `C`, durante os testes da variante `<X.002,XX.YY.ZZ>`, as possíveis candidatas a receber a propagação seriam as variantes

<X.003,XX.YY.ZZ> e <X.004,XX.YY.ZZ>, pois possuem a mesma característica implementada. Entretanto, devido a falta de outros documentos que permitam confirmar a reprodutibilidade do erro, se faz necessário executar testes e inspecionar visualmente o código dessas variantes antes de efetuar a propagação

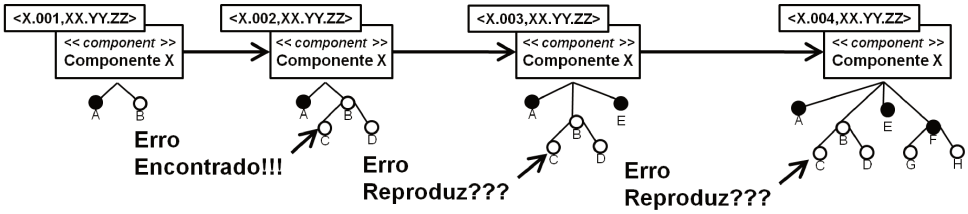

Figura 4.4: Propagação de correções de erros entre variantes do *Componente X*.

Motivação

A principal motivação para a propagação da correção de erros é a necessidade de se manter estáveis as múltiplas variantes de um componente e evitar que erros conhecidos sejam reportados novamente em um ciclo de testes de outra variante.

Conseqüências

O objetivo da atividade de correção de erros é fazer com que uma característica funcione conforme a sua especificação. Por isso, não é esperado que ocorra impacto nos documentos de requisitos. Entretanto, uma correção pode vir a alterar a arquitetura da linha de produto ou o projeto do componente em que ela será implementada. Nestes casos, os documentos correspondentes devem ser atualizados e as recomendações apresentadas no cenário *Criar ou manter múltiplas variantes de um componente em paralelo* (Seção 4.2.1) devem ser seguidas.

O principal risco para as variantes de um componente que recebem uma propagação está na possibilidade de ocorrer algum efeito colateral. Por isso, em alguns casos, uma correção não é propagada devido ao alto risco da implementação perante a severidade do erro.

Relacionamento com outros padrões de evolução

Este cenário é uma conseqüência direta da opção pela manutenção de múltiplas variantes de um componente em paralelo(seções 4.2.1 e 4.2.2).

4.2.4 Convergir várias variantes de um componente criando uma nova variante

Descrição

Conforme apresentado no cenário *Criar ou manter múltiplas variantes de um componente em paralelo* (Seção 4.2.1), as variantes de um componente podem evoluir separadamente. Por isso, o conjunto de características disponíveis estará distribuído ao longo de várias variantes. Em alguns casos, pode-se ter uma característica implementada por todas as variantes, por um subconjunto delas ou apenas em uma variante. A Figura 4.5 baseia-se na evolução do componente `Componente X` para ilustrar esse ambiente de desenvolvimento. Neste contexto, uma determinada linha de produto pode requisitar características que estão implementadas em diferentes variantes. Assim, a abordagem de convergência de variantes é uma alternativa adequada.

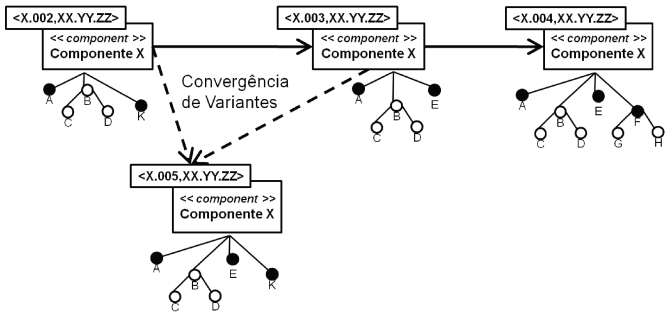

Figura 4.5: Convergência de variantes do *Componente X*.

Nesta abordagem de evolução, duas ou mais variantes de um componente servirão como base para a criação de uma nova. As variantes de origem podem ser substituídas ou mantidas em paralelo com a variante criada. No primeiro caso, a aplicação sucessiva deste cenário de evolução pode fazer com que exista uma única variante do componente sendo compartilhada entre as linhas de produto. A variante criada através desta abordagem de evolução será um super conjunto das variantes de origem, em termos do conjunto de características disponíveis.

Ainda na Figura 4.5, pode-se ver que a variante `<X.002,XX.YY.ZZ>` sofreu uma evolução após servir de base para a criação da variante `<X.003,XX.YY.ZZ>`, pois a característica `K` está presente apenas em `<X.002,XX.YY.ZZ>`. Entretanto, uma nova linha de produtos será criada e deseja o conjunto de características atuais das variantes `<X.002,XX.YY.ZZ>` e `<X.003,XX.YY.ZZ>`. Assim, a variante `<X.005,XX.YY.ZZ>` foi criada

tendo como base ambas as variantes e possuindo um super conjunto de suas características (a `Característica E` da variante `<X.003,XX.YY.ZZ>` e a característica `K` da variante `<X.002,XX.YY.ZZ>`). Neste exemplo, todas as três variantes continuarão a evoluir em paralelo.

Motivação

Os seguintes fatores podem motivar a criação de uma nova variante de um componente a partir de múltiplas variantes existentes:

- Unificação do conjunto de características disponíveis.
- Redução do custo associado a propagações de erros (Seção 4.2.3), características (Seção 4.2.5) e testes se as variantes que deram origem ao componente forem descontinuadas.
- Unificação dos artefatos de software associados ao componente.
- Unificação de melhorias arquiteturais ou de projeto realizadas de modo separado em cada variante.

Conseqüências

Os seguintes artefatos devem ser alterados ou criados após a criação de um componente baseado em variantes existentes:

- *Requisitos:* os documentos utilizados na especificação de requisitos devem ser analisados cuidadosamente para se identificar requisitos conflitantes. Após esta análise, um documento único deve ser gerado para a nova variante.
- *Arquitetura Baseada em Componentes:* a arquitetura de cada variante do componente deve ser analisada de modo a elaborar uma nova arquitetura que seja capaz de satisfazer os requisitos de qualidade de ambas as variantes. Além disso, as interfaces de cada variante devem ser unificadas.
- *Projeto "Detalhado":* a implementação de uma determinada característica pode ter sido realizada de forma diferente nas variantes de origem. O documento de projeto da nova variante deve unificar essas diferenças de forma a garantir a implementação mais adequada.

Deve-se analisar a existência de dependências externas, uma vez que é necessário garantir que elas existam no mesmo ambiente onde a nova variante irá ser utilizada.

Relacionamento com outros padrões de evolução

Esta abordagem de evolução pode ser considerada o oposto e também uma extensão do proposto no cenário *Criar ou manter múltiplas variantes de um componente em paralelo* (Seção 4.2.1). Quando as variantes de origem são descontinuadas, pode-se dizer que o número de variantes simultâneas existentes está sendo reduzido. Entretanto, se elas continuam a ser mantidas, tem-se apenas mais uma forma de gerar novas variantes de um componente.

A convergência de variantes pode ainda ser vista como uma seqüência de aplicações do cenário *Incluir característica de uma variante em outra do mesmo componente* (Seção 4.2.5). Entretanto, a principal diferença está na possibilidade de existirem múltiplas variantes de origem e no fato de uma nova variante ser criada, ao invés de haver uma troca de características entre duas variantes.

4.2.5 Incluir característica de uma variante em outra do mesmo componente

Descrição

Nesta abordagem de evolução, uma linha de produto que utiliza uma determinada variante de um componente faz uma requisição de uma característica disponível em outra variante. Neste caso, não se pode simplesmente substituir as variantes utilizadas, pois a variante corrente pode conter características não existentes na variante onde a característica desejada está presente. Por isso, deve-se evoluir a variante atual para incluir a nova característica.

A Figura 4.6 ilustra a inclusão da característica `F` e suas características opcionais disponibilizados na variante `<X.004,XX.YY.ZZ>` do componente `Componente X` para a variante `<X.005,02.YY.ZZ>`. Note que, após a inclusão, apenas uma revisão é realizada (representada pela mudança no identificador de versão da variante de XX.YY.ZZ (Seção 4.2.4) para 02.YY.ZZ). Neste padrão, uma nova variante não é criada, mas sim uma revisão da variante atual com a inclusão da nova característica.

Motivação

O custo da inclusão de uma característica existente em outra variante do mesmo componente deve ser comparado ao custo de se escrever a mesma característica desde o início. Esta análise determinará se esta abordagem de evolução é válida. No contexto deste trabalho, as várias variantes de um componente derivam de uma base comum e possuem

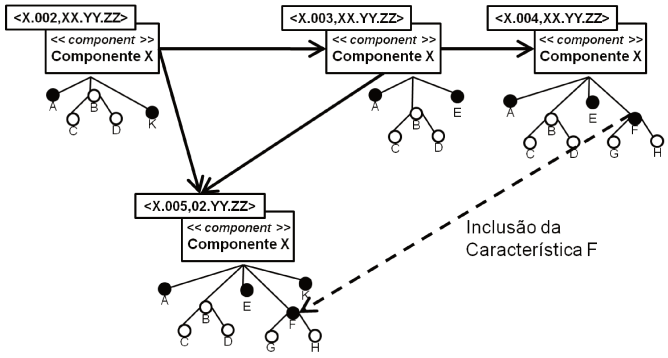

Figura 4.6: Inclusão da *Característica F* da variante *04.YY.ZZ* para a *05.YY.ZZ*.

muitas similaridades. Assim, este padrão de evolução torna-se a melhor alternativa na maioria dos casos.

A requisição de uma característica existente pode ocorrer devido a identificação de que ela trouxe uma vantagem competitiva para a linha de produto que a utilizava.

Conseqüências

O impacto nos artefatos associados ao componente quando é realizada uma inclusão de característica é semelhante ao do processo de criação. Entretanto, os artefatos existentes podem ser modificados tendo como base as alterações já realizadas na variante de origem. Quando isto não é possível devido a diferenças de arquitetura e projeto na variante corrente, as dicas apresentadas no padrão *Convergir várias variantes de um componente criando uma nova variante* (Seção 4.2.4) devem ser seguidas.

Relacionamento com outros padrões de evolução

Este cenário de evolução se relaciona com o padrão *Propagar correção de erros entre variantes de um componente* (Seção 4.2.3), pois os dois envolvem a inclusão de alterações entre duas variantes de um componente sem gerar uma nova variante. Entretanto, a propagação da correção de erros tem apenas o objetivo de fazer com que uma característica funcione como esperado, geralmente envolve poucas linhas de código e têm um impacto baixo nos demais documentos. Enquanto isso, a inclusão de características envolve uma mudança de comportamento que pode impactar a arquitetura e o projeto de um componente.

A inclusão de características de uma variante em outra do mesmo componente pode ser vista como um passo na aplicação do padrão de evolução *Convergir várias variantes de um componente criando uma nova variante* (Seção 4.2.4).

4.3 Análise da Família de Padrões de Evolução na empresa Motorola Inc

O objetivo desta seção é ilustrar, com exemplos reais, o uso dos padrões de evolução apresentados na Seção 4.2. Para isso, foram utilizadas informações da Motorola, empresa dos Estados Unidos da América (EUA) com foco no desenvolvimento de celulares, infra-estrutura de redes e em soluções de mobilidade para os lares. No entanto, por motivos de confidencialidade, não é possível divulgar os nomes dos projetos, produtos ou clientes relacionados às linhas de produto. Este estudo foi realizado a partir da análise da evolução dos componentes `Instant Messaging` e `Messaging` mantidos pela empresa e que são compartilhados por várias arquiteturas de linhas de produtos voltados ao desenvolvimento de celulares.

A Seção 4.3.1 apresenta a análise da evolução do componente `Instant Messaging` e a Seção 4.3.2 descreve o resultado da mesma análise para o componente `Messaging`. Por fim, a Seção 4.3.3 apresenta algumas considerações sobre os dados apresentados e outras informações sobre a evolução de componentes nas linhas de produtos da Motorola.

4.3.1 O Componente Instant Messaging

A evolução do componente `Instant Messaging`, responsável por prover serviços de comunicação instantânea aos celulares comercializados pela Motorola, é analisada neste estudo de caso. A Figura 4.7 ilustra as variantes do componente criadas no período de outubro de 2004 a outubro de 2008, onde as setas tracejadas indicam quais arquiteturas de linhas de produto usam uma determinada variante do componente. Por exemplo, a `ALP F` utiliza a variante `<IM.004>`. As setas cheias indicam a origem de uma determinada variante, como no caso em que a variante `<IM.008>` serve de base para a criação da variante `<IM.011>`. As setas pontilhadas informam a ocorrência de inclusões de características entre as variantes. Apenas os identificadores das variantes são apresentados para simplificar a figura. Neste momento, a informação do porquê da criação das variantes e linhas de produto não é relevante.

A Figura 4.8 ilustra o paralelismo existente entre as variantes do componente `Instant Messaging` da Figura 4.7. As linhas em negrito indicam o ponto de criação e término de cada variante, onde as linhas de produtos que utilizavam as variantes foram descontinuadas. Analisando as Figuras 4.7 e 4.8, pode-se notar alguns dos padrões de evolução apresentados na Seção 4.2, como: (i) a criação das variantes `<IM.006>`, `<IM.007>` e `<IM.009>` a partir da `<IM.005>` e o paralelismo existente entre as diversas variantes são exemplos

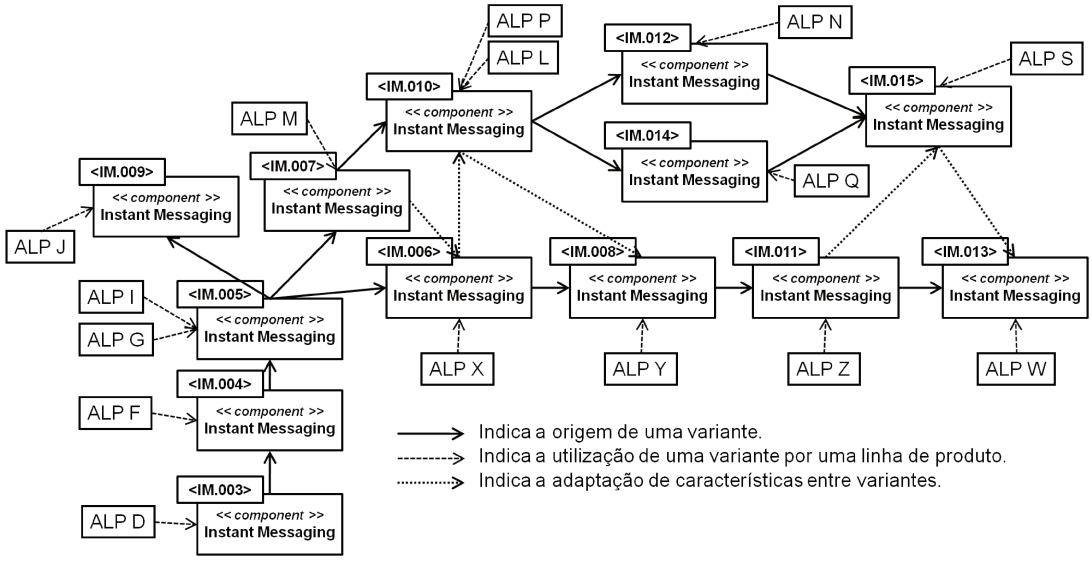

Figura 4.7: Evolução do componente *Instant Messaging*.

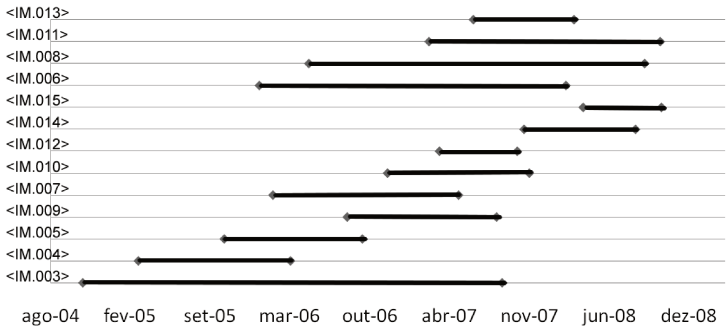

Figura 4.8: Paralelismo das variantes do componente *Instant Messaging*.

de aplicações do padrão *Criar ou manter múltiplas variantes de um componente em paralelo*; (ii) a criação da variante `<IM.015>` a partir das variantes `<IM.012>` e `<IM.014>` é um exemplo do padrão *Convergir várias variantes de um componente criando uma nova variante*; (iii) a inclusão de características da variante `<IM.006>` para a `<IM.010>` exemplifica a aplicação do padrão *Incluir característica de uma variante em outra do mesmo componente*; (iv) o compartilhamento da variante `<IM.010>` entre a `ALP P` e a `ALP L` ilustra o padrão *Manter uma única variante de um componente.*

A Tabela 4.1 apresenta uma análise quantitativa da evolução do componente `Instant Messaging`, onde a coluna `Padrões de Evolução` mostra os padrões de evolução descritos na Seção 4.2 e a coluna `Número de Ocorrências` descreve o período e a quantidade de vezes que cada padrão ocorreu nas linhas de produto que utilizavam o componente. A Figura

4.8 mostra a existência de um alto grau de paralelismo entre as variantes do componente `Instant Messaging`, o que sugere uma grande ocorrência de propagações de erros entre elas. Este paralelismo pode ser comprovado pela análise quantitativa do padrão III apresentada na Tabela 4.1.

	Padrões de Evolução	Número de Ocorrências
I	***Criar múltiplas variantes de um componente em paralelo***	Durante o período de agosto de 2004 a dezembro de 2008, ocorreram 25 casos de manutenção de variantes simultâneas do componente *Instant Messaging* pela Motorola. Apenas no mês de março de 2006, existiam 6 variantes sendo mantidas simultaneamente.
II	***Criar múltiplas variantes temporárias de um componente***	No período de setembro de 2005 a dezembro de 2008, a partir dos dados de 8 linhas de produto, houve a criação de 59 variantes temporárias do componente *Instant Messaging*.
III	***Propagar correção de erros entre variantes de um componente***	Durante o período de setembro de 2007 a dezembro de 2008, foram abertas 710 requisições de mudança devido a erros no componente *Instant Messaging*. Dentre estas mudanças, houve a necessidade de se realizar 209 propagações entre as diferentes variantes ativas no período.
IV	***Convergir várias variantes de um componente criando uma nova variante***	Durante o período de agosto de 2004 a dezembro de 2008, ocorreu apenas um caso de convergência de variantes do componente *Instant Messaging*. Neste exemplo, apenas duas variantes foram usadas como base e elas não foram descontinuadas.
V	***Incluir característica de uma variante em outra do mesmo componente***	No período de agosto de 2004 a dezembro de 2008, ocorreram 36 casos de adaptações de funcionalidades entre variantes do componente *Instant Messaging*.

Tabela 4.1: Análise estatística do componente *Instant Messaging*.

O padrão *Criar ou manter múltiplas variantes temporárias de um componente* é exemplificado na Figura 4.9, onde três variantes temporárias do componente `Instant Messaging` (`<IM.012_A>`, `<IM.012_B>` e `<IM.012_C>`) foram criadas a partir da variante principal `<IM.012>`, utilizada pela `ALP N`, para atender aos clientes de diferentes regiões. Neste exemplo, apesar de todas as variantes temporárias derivarem da variante `<IM.012>`, cada uma é derivada a partir de uma determinada versão desta variante. A ocorrência deste padrão também pode ser comprovada através do resultado da análise quantitativa do padrão II na Tabela 4.1.

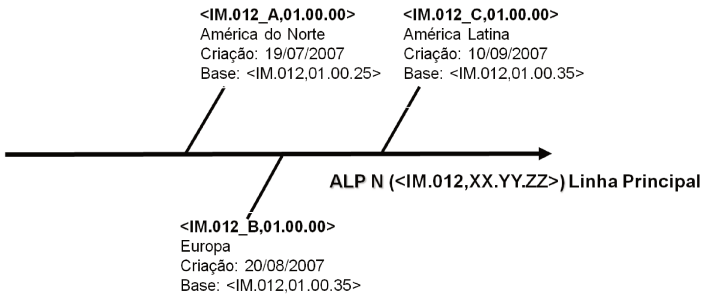

Figura 4.9: Variantes temporárias do componente *Instant Messaging*.

4.3.2 O Componente Messaging

A evolução do componente `Messaging`, responsável por prover serviços de mensagens de texto, mensagens multimídia e correio de voz aos celulares comercializados pela Motorola, é analisada neste estudo de caso. A Figura 4.10 ilustra as variantes do componente criadas no período de dezembro de 2003 a outubro de 2008, onde as setas tracejadas indicam quais arquiteturas de linhas de produto usam uma determinada variante do componente. Por exemplo, a `ALP E` utiliza a variante `<MSG.004>`. As setas cheias indicam a origem de uma determinada variante, como no caso em que a variante `<MSG.011>` serve de base para a criação da variante `<MSG.012>`. As setas pontilhadas informam a ocorrência de inclusões de características entre as variantes. Apenas os identificadores das variantes são apresentados para simplificar a figura. Neste momento, a informação do porquê da criação das variantes e linhas de produto não é relevante.

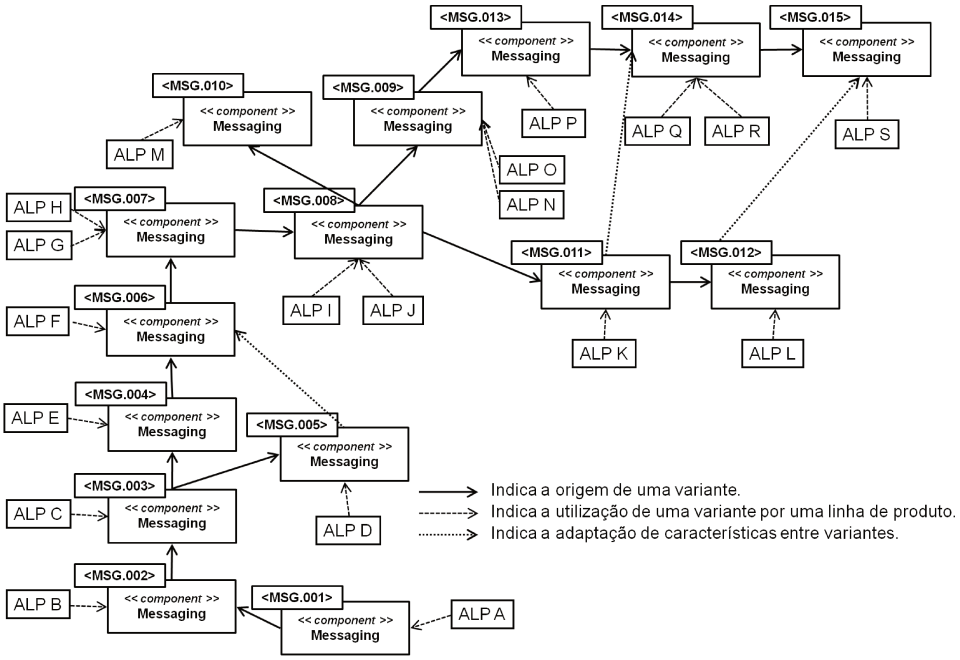

Figura 4.10: Evolução do componente *Messaging*.

A Figura 4.11 ilustra o paralelismo existente entre as variantes do componente `Messaging` da Figura 4.10. As linhas em negrito indicam o ponto de criação e término de cada variante, onde as linhas de produtos que utilizavam as variantes foram descontinuadas. Analisando as Figuras 4.10 e 4.11, pode-se notar alguns dos padrões de evolução apresentados na Seção 4.2, como: (i) a criação das variantes `<MSG.010>`, `<MSG.009>` e `<MSG.011>` a partir da `<MSG.008>` e o paralelismo existente entre as diversas variantes são exemplos de

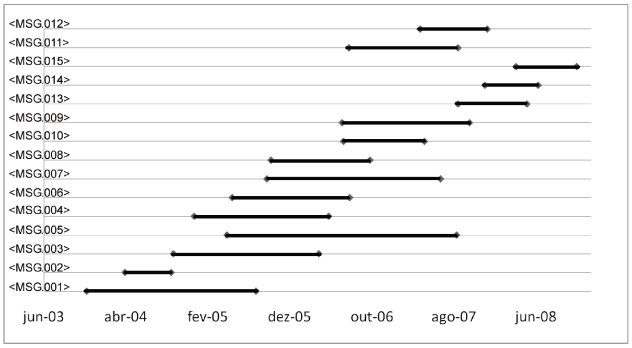

Figura 4.11: Paralelismo das variantes do componente *Messaging*.

aplicações do padrão *Criar ou manter múltiplas variantes de um componente em paralelo*; (ii) a inclusão de características da variante `<MSG.012>` para a `<MSG.015>` exemplifica a aplicação do padrão *Incluir característica de uma variante em outra do mesmo componente*; (iii) o compartilhamento da variante `<MSG.007>` entre a `ALP H` e a `ALP G` ilustra o padrão *Manter uma única variante de um componente*.

A Tabela 4.2, que possui formato análogo à Tabela 4.1, apresenta uma análise quantitativa da evolução do componente `Messaging` no período de dezembro de 2003 a outubro de 2008. A Figura 4.11 mostra a existência de um alto grau de paralelismo entre as variantes do componente `Messaging`, o que sugere uma grande ocorrência de propagações de erros entre elas. Este paralelismo pode ser comprovado pela análise quantitativa do padrão III apresentada na Tabela 4.2.

	Padrões de Evolução	Número de Ocorrências
I	***Criar múltiplas variantes de um componente em paralelo***	Ocorreram 37 casos de manutenção de variantes simultâneas do componente *Messaging* pela Motorola. Apenas no mês de dezembro de 2005, existiam 6 variantes sendo mantidas simultaneamente.
II	***Criar múltiplas variantes temporárias de um componente***	A partir dos dados de 18 linhas de produto, houve a criação de 183 variantes temporárias do componente *Messaging*.
III	***Propagar correção de erros entre variantes de um componente***	Foram abertas 2836 requisições de mudança devido a erros no componente *Messaging*. Dentre estas mudanças, houve a necessidade de se realizar 1023 propagações entre as diferentes variantes ativas no período.
IV	***Convergir várias variantes de um componente criando uma nova variante***	Não ocorreram casos de convergência de variantes do componente *Messaging*.
V	***Incluir característica de uma variante em outra do mesmo componente***	Ocorreram 87 casos de adaptações de funcionalidades entre variantes do componente *Messaging*.

Tabela 4.2: Análise estatística da evolução no componente *Messaging*.

4.3.3 Análise dos Exemplos da Motorola

Os diversos componentes evoluem de forma independente um do outro. Por exemplo, na Figura 4.7 pode-se notar que a variante `<IM.010>` do componente `Instant Messaging` é utilizada pela `ALP P` e `ALP L`. Entretanto, no caso do componente `Messaging`, a `ALP P` utiliza a variante `<MSG.013>` e a `ALP L` utiliza a variante `<MSG.012>`. Isto indica que a criação de variantes depende do conjunto de características requisitado e da decisão de cada gerente de componente. Além disso, um componente não precisa ser utilizado em todas as linhas de produto existentes. Por exemplo, na Figura 4.10 o componente `Messaging` está presente na `ALP R`, mas essa mesma arquitetura não utiliza o componente `Instant Messaging`.

Nas linhas de produto da Motorola, a variabilidade não é implementada apenas através do uso de variantes de componentes. Para implementar características opcionais ou alternativas, diretivas de compilação e parâmetros de configuração também são utilizados. No caso de diretivas de compilação, quando uma característica não é selecionada, o código é completamente removido e libera-se espaço de memória no telefone. Entretanto, a resolução deste ponto de variação deve ocorrer em tempo de fabricação dos celulares. No caso de parâmetros de configuração, os clientes tem a possibilidade de configurarem as características manualmente. Esta abordagem permite que os próprios clientes adaptem os celulares para cada região onde eles serão comercializados ou realizem modificações automaticamente via a própria rede celular. Neste caso, a característica estará presente no telefone, mas pode estar habilitada para um conjunto de celulares e desabilitada para outros. Além disso, uma característica pode ser implementada através da criação de uma nova variante, estar sob diretiva de configuração e ser adaptado através de parâmetros de configuração ao mesmo tempo. De uma forma geral, este cenário ocorre quando a característica implementada é opcional ou alternativa dentro da própria linha de produto que a solicitou. O gerente da linha de produto é o responsável por resolver todos os pontos de variação disponibilizados pelos componentes.

Uma outra característica interessante dos componentes mantidos pela Motorola e que foi identificada como um padrão na indústria por `Bosch et al` [9], está relacionada ao número de linhas de código de um componente. Por exemplo, o número de linhas do componente `Instant Messaging` é de 183.890 e do componente `Messaging` é de 319.213.

A identificação desta família de padrões de evolução apresenta dois benefícios imediatos:

- Estabelece uma linguagem comum entre os gerentes de componente e gerentes de

linha de produto. No caso da Motorola, esses dois tipos de gerentes são numerosos e localizados em diversos países. Por isso, a família de padrões pode contribuir para um melhor entendimento entre eles; e

- Documenta cenários de evolução recorrentes permitindo aos gerentes de linhas de produtos antecipar eventuais impactos que eles possam causar nas arquiteturas de linhas de produto.

Capítulo 5

Estudo de Caso: Linha de Produto MobileMedia

Este estudo de caso tem como objetivo simular um ambiente em que múltiplas linhas de produto de software são mantidas em paralelo. Desta forma, os padrões identificados (Capítulo 4) e a tabela de relacionamento entre características e variantes de componentes (Seção 3.3) podem ser validados. Para isto, a linha de produto de software conhecida por MobileMedia foi utilizada.

O MobileMedia é uma linha de produto de software para a geração de aplicações para o gerenciamento de arquivos de áudio, vídeo ou imagem em dispositivos móveis. Ela foi desenvolvida por pesquisadores da Lancaster University [15], que estenderam a versão original, chamada de MobilePhoto [38], de forma a adicionar novas características mandatórias, opcionais e alternativas. Após este trabalho, ela foi adaptada ao modelo de componentes COSMOS* [14] [17] pelo Software Engineering and Dependability Research Group, um grupo de pesquisa situado no Instituto de Computação da Unicamp.

A Figura 5.1 apresenta o modelo de características completo da linha de produto do MobileMedia. Neste modelo, os três tipos de mídia apoiados podem ser identificados: *Imagem*, *Áudio* e *Vídeo*. Estas características são implementadas como alternativas do tipo ou-inclusivas, ou seja, uma ou mais podem ser selecionadas. As características mandatórias são *Gerenciamento de Álbuns* e *Operações Básicas*. As características opcionais são: *Favoritos*, *Ordenar*, *Copiar Mídia*, *Transferir via SMS*, *Capturar Imagem* e *Capturar Vídeo*. As características mandatórias estão disponíveis para qualquer dispositivo com suporte a plataforma J2ME.

O componente `MobileMedia` é o responsável por implementar o modelo de carac-

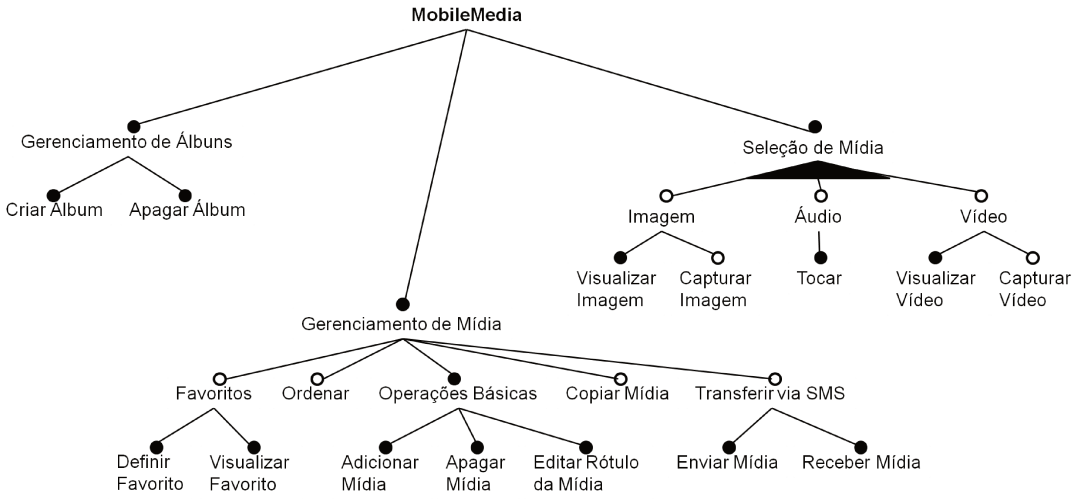

Figura 5.1: Modelo de características da linha de produto do MobileMedia.

terísticas apresentado na Figura 5.1. A Figura 5.2 apresenta um modelo parcial da arquitetura em que o componente é utilizado. Nesta arquitetura, o ponto de variação `<PV_MM.01.00.00>` representa a possibilidade de se utilizar diferentes variantes do componente `MobileMedia`. Além dele, outros três componentes fixos estão presentes: `FileSystemMgr`, `ExceptionHandler` e `MobileResources`. As variantes do componente `MobileMedia` que podem ser associadas ao ponto de variação existente são ilustradas na Figura 5.3.

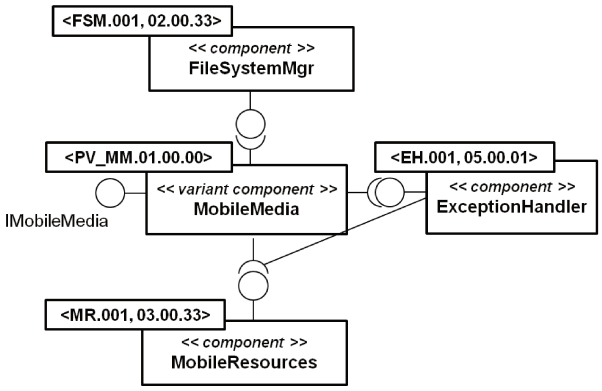

Figura 5.2: Modelo parcial da arquitetura de linha de produto em que o componente *MobileMedia* é utilizado.

A Tabela 5.1 apresenta a evolução da linha de produto de software do MobileMedia, com uma breve descrição das modificações que deram origem as oito variantes existentes.

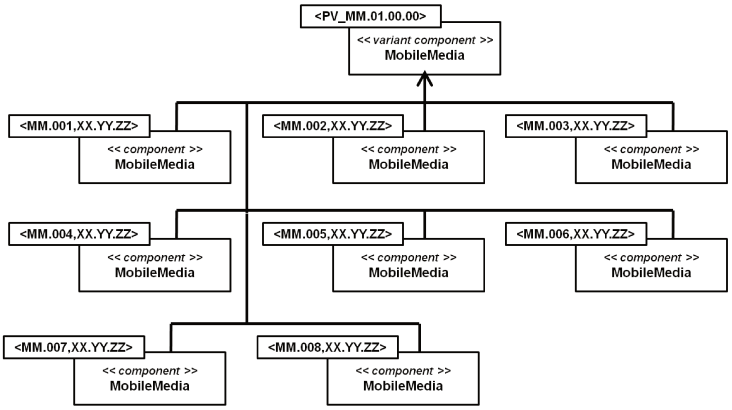

Figura 5.3: Variantes disponíveis para o componente *MobileMedia*.

A primeira variante é equivalente ao original MobilePhoto e a demais foram criadas através da inclusão de novas características. Dentre estas, apenas a variante original não foi disponibilizada seguindo o modelo COSMOS*.

Para cada variante do MobileMedia apresentada na Tabela 5.1, existem duas técnicas distintas para o tratamento da variabilidade, uma baseada em orientação a aspectos [21] e a outra em orientação a objetos [16]. Neste estudo de caso, apenas as variantes implementadas de acordo com as técnicas de orientação a objetos serão consideradas. Nesta abordagem, a variabilidade foi implementada através do uso de diretivas de compilação e padrões de projeto.

A Tabela 5.2 apresenta a relação entre as características disponíveis para a linha de produto do MobileMedia (Figura 5.1) e as variantes do componente `MobileMedia` (Figura 5.3) que as implementam. Assim, conforme discutido na Seção 3.3, o vetor de características da linha de produto do MobileMedia é `[Gerenciamento de Álbuns`, `Imagem`, `Áudio`, `Vídeo`, `Capturar Imagem`, `Capturar Vídeo`, `Favoritos`, `Ordenar`, `Adicionar Mídia`, `Apagar Mídia`, `Editar Rótulo da Mídia`, `Copiar Mídia`, `Transferir via SMS]`. Como exemplo, caso alguma arquitetura de linha de produto precise da característica relacionada a captura de vídeo, a variante `<MM.008>` do componente `MobileMedia` deverá ser escolhida. A efetividade da tabela de relacionamento entre características e variantes proposta na Seção 3.3 pode ser comprovada neste estudo de caso. Apesar de ser uma solução simples, ela apresenta de forma muito prática os dados necessários para auxiliar os gerentes de linhas de produto a instanciar os produtos a serem produzidos.

O componente `MobileMedia` é composto por vários outros subcomponentes. Em uma linha de produto de software, o grupo responsável pelo desenvolvimento dos produtos não

Variante	Descrição da Alteração	Tipo da Alteração
<MM.001>	Implementação original do MobilePhoto. Nesta versão é possível gerenciar álbuns (apagar e criar) e imagens (adicionar, apagar e visualizar).	Implementação inicial.
<MM.002>	Inclusão de suporte ao tratamento de exceções.	Inclusão de uma característica não-funcional obrigatória.
<MM.003>	Duas novas característica foram adicionadas: • contagem do número de visualizações de uma imagem, permitindo a ordenação pela quantidade de vezes que as imagens foram acessadas. • edição do rótulo de uma imagem.	Inclusão de uma característica opcional (ordenação) e outra mandatória (edição do rótulo).
<MM.004>	Nova característica adicionada para permitir aos usuários especificar e visualizar imagens favoritas.	Inclusão de uma característica opcional.
<MM.005>	Suporte para a manutenção de múltiplas cópias de imagens.	Inclusão de uma característica opcional.
<MM.006>	Nova característica adicionada para enviar e receber imagens através de mensagens de texto.	Inclusão de uma característica opcional.
<MM.007>	Nova característica para o gerenciamento de arquivos de áudio. O gerenciamento de imagens tornou-se uma característica alternativa. Todas as características opcionais também foram disponibilizadas para áudio.	Alteração de uma característica mandatória em duas alternativas.
<MM.008>	Suporte para o gerenciamento de vídeo, incluindo as características opcionais existentes para áudio e imagem. Além disso, foi adicionada a possibilidade de capturar imagens e vídeos.	Inclusão de uma característica alternativa (gerenciamento de vídeo) e duas opcionais (captura de imagem e vídeo).

Tabela 5.1: Evolução da Linha de Produto MobileMedia.

precisa conhecer a arquitetura interna de um componente. O importante é conhecer o modelo de características associado ao gerenciamento de mídia e indicar na arquitetura da linha de produto um único componente responsável por implementá-las. A arquitetura interna é de interesse apenas do grupo responsável por manter o componente entre os ativos centrais [10]. Neste estudo de caso, o versionamento ocorreu no nível do componente `MobileMedia`, ou seja, seus subcomponentes não são versionados.

As variantes em COSMOS* do MobileMedia são utilizadas para simular o ambiente em que múltiplas linhas de produto de software são mantidas em paralelo. A arquitetura das linhas de produto é a mesma apresentada na Figura 5.2 e o componente `MobileMedia` será compartilhado entre elas. A partir da montagem deste ambiente simulado, a aplicação de alguns padrões definidos na família de padrões de evolução do Capítulo 4 foi realizada. Desta forma, o impacto de cada padrão de evolução no componente pode ser avaliado de forma mais precisa, auxiliando na identificação de novas características e na validação das informações encontradas pela análise de dados da Motorola.

A Figura 5.4 apresenta o início onde apenas uma arquitetura de linha de produto (`ALP A`) utilizava o componente `MobileMedia`. Neste ponto, apenas a variante `<MM.001>` estava disponível e, como ilustra o modelo de características na mesma figura, apenas o

Variantes	Características												
	Gerenciamento de Álbuns	Imagem	Áudio	Vídeo	Capturar Imagem	Capturar Vídeo	Favoritos	Ordenar	Adicionar Mídia	Apagar Mídia	Editar Rótulo da Mídia	Copiar Mídia	Transferir via SMS
<MM.001>	X	X							X	X			
<MM.002>	X	X						X	X	X	X		
<MM.003>	X	X					X	X	X	X	X		
<MM.004>	X	X					X	X	X	X	X	X	
<MM.005>	X	X					X	X	X	X	X	X	X
<MM.006>	X	X					X	X	X	X	X	X	X
<MM.007>	X	X	X				X	X	X	X	X	X	X
<MM.008>	X	X	X	X	X	X	X	X	X	X	X	X	X

Tabela 5.2: Relacionamento entre as características disponíveis na linha de produto do MobileMedia e as variantes do componente *MobileMedia*.

gerenciamento básico de imagens era necessário.

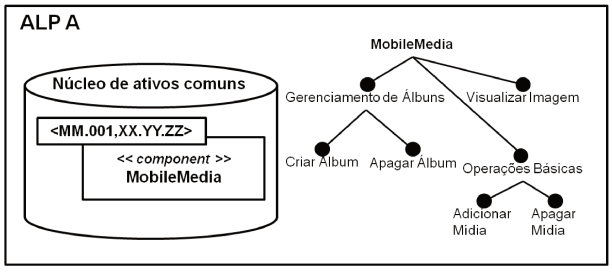

Figura 5.4: Representação do contexto em que apenas uma linha de produto utiliza o componente *MobileMedia*.

A Figura 5.5 ilustra o ponto em que uma nova linha de produtos (representada pela arquitetura de linha de produto `ALP B`) é criada e decide utilizar o componente `MobileMedia`, mas requisita novas características: Ordenar e Editar Rótulo de Mídia. Estas novas requisições são implementadas através da criação de uma nova variante do componente, com a característica opcional de ordenação disponibilizada com o uso de diretivas de compilação. Assim, o componente `MobileMedia` passa a possuir duas variantes (`<MM.001>` e `<MM.002>`) que podem ser escolhidas dependendo do conjunto de características necessário por cada linha de produto. A partir deste momento, o componente `MobileMedia` passa a fazer parte dos ativos centrais comuns entre ambas linhas de produtos. Esta abordagem representa a adoção do padrão de evolução *Criar ou manter múltiplas variantes de um componente em paralelo* (Seção 4.2.1). Conforme ilustrado na árvore de relacionamento da Figura 3.3 (Seção 3.2), a opção por este padrão de evolução pode significar a adoção

dos outros padrões apresentados no Capítulo 4 nas próximas vezes em que o componente evoluir.

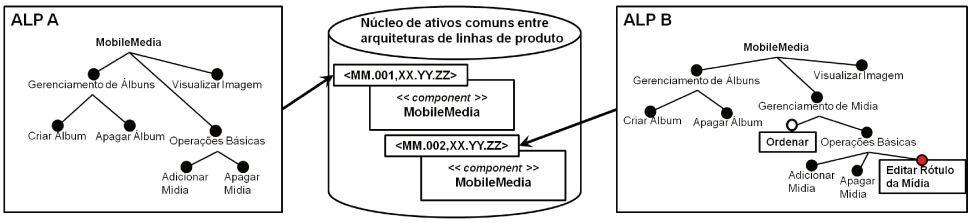

Figura 5.5: Representação do contexto em que duas linhas de produto utilizam o componente *MobileMedia*.

Com a criação de outras linhas de produto e a necessidade de adição de novas características, novas versões do componente `MobileMedia` foram disponibilizadas. O relacionamento entre estas versões e o modelo de características foi mapeado na Tabela 5.2.

Além do padrão *Criar ou manter múltiplas variantes de um componente em paralelo* (Seção 4.2.1), o padrão *Propagar correção de erros entre variantes de um componente* (Seção 4.2.3) também foi utilizado neste estudo de caso. A sua utilização ocorreu quando erros foram encontrados durante o teste de características em variantes mais recentes do componente `MobileMedia`. Após realizada a correção, ela foi propagada para as variantes iniciais do componente, quando aplicável.

Através deste estudo de caso, pode-se validar e compreender melhor as características e consequências dos padrões de evolução *Criar ou manter múltiplas variantes de um componente em paralelo* (Seção 4.2.1) e *Propagar correção de erros entre variantes de um componente* (Seção 4.2.3).

Capítulo 6

Conclusões

Este capítulo apresenta as conclusões e contribuições deste trabalho (Seção 6.1) e as possibilidades de trabalhos futuros (Seção 6.2).

6.1 Conclusões e Contribuições

O gerenciamento da evolução em uma linha de produto de software pode ser complexo, pois alterações realizadas em um único componente podem impactar os demais ativos relacionados e produtos desenvolvidos. Esta dissertação explora a evolução em componentes que são compartilhados por múltiplas linhas de produto de software mantidas em paralelo. Neste contexto, um componente é utilizado por duas ou mais arquiteturas de linhas de produto ao mesmo tempo, podendo sofrer requisições de mudanças relativas à evolução do modelo de características de cada uma das linhas.

Uma família de cinco padrões de evolução em componentes compartilhados por múltiplas linhas de produto de software foi apresentada, com o objetivo de descrever o contexto em que cada padrão é aplicável, as vantagens e os riscos associados ao seu uso. Os padrões descritos foram:

- *Criar ou manter múltiplas variantes de um componente em paralelo*: uma única variante de um componente pode dar origem a uma ou mais variantes que serão mantidas em paralelo à variante original. Estas novas variantes possuem ciclo de vida próprio e poderão dar origem a outras.

- *Criar ou manter múltiplas variantes temporárias de um componente*: uma variante de referência para um componente é definida e novas variantes são temporariamente derivadas para atender a um demanda específica. A variante de referência sem-

pre representará um superconjunto das características de todas as suas variantes derivadas.

- *Propagar correção de erros entre variantes de um componente*: quando uma correção de erro é realizada em uma determinada variante de um componente, deve-se analisar a necessidade de propagação da correção para as demais variantes existentes.
- *Convergir várias variantes de um componente criando uma nova variante*: uma variante de um componente é criada a partir de duas ou mais variantes existentes.
- *Incluir característica de uma variante em outra do mesmo componente*: uma linha de produto que utiliza uma determinada variante de um componente faz uma requisição de uma característica disponível apenas em outra variante do mesmo componente.

Esta família de padrões de evolução foi identificada através de estudos realizados em linhas de produto de software reais para celulares mantidas pela Motorola. A identificação destes padrões permite estabelecer uma linguagem comum entre os responsáveis pela evolução dos componentes e linhas de produto. Além disso, contribui para que a evolução ocorra de forma controlada e sistemática. Contudo, este estudo limita-se ao contexto da Motorola e, portanto, é necessário avaliar se essa família de padrões ocorre em outras empresas e identificar potenciais novos padrões.

A escolha do padrão de evolução a ser utilizado, quando uma requisição de mudança de característica é realizada, é apoiada por um processo simplificado que apresenta um conjunto de atividades associadas aos responsáveis pela linha de produto e pelos componentes existentes. O fluxograma sugerido auxilia no processo de tomada de decisões que os gerentes de linhas de produto e componentes devem executar ao longo da evolução das características apoiadas inicialmente.

Para apoiar a escolha da variante de um componente a ser utilizada por uma arquitetura de linha de produto, uma tabela é apresentada como ferramenta para mapear as variantes dos componentes às características existentes. Esta tabela consolida as informações sobre quais características estão disponíveis para um gerente de linhas de produto e qual variante do componente desejado as disponibiliza. Desta forma, o gerente de linha de produto pode usar uma variante existente ou solicitar ao gerente de componente que uma determinada versão contenha alguma característica específica já existente em outra variante. O gerente do componente será responsável por decidir o melhor padrão de evolução, podendo criar uma nova variante.

O uso do processo de apoio a evolução e o conhecimento dos padrões de evolução auxilia os gerentes de linhas de produto e de componentes a gerenciar de forma mais controlada e sistemática a evolução das características associadas as suas linhas de produto.

As principais contribuições deste trabalho foram:

- Família de padrões de evolução;
- Tabela de mapeamento que relaciona as características existentes às variantes disponíveis de um componente e o processo para a sua utilização ao se instanciar a arquitetura da linha de produto;
- Processo para auxiliar na análise do padrão de evolução a ser adotado para se implementar uma requisição de mudança de característica.

6.2 Trabalhos Futuros

Como trabalhos futuros, a partir da família de padrões apresentada, pode-se:

- definir diretrizes para auxiliar na implementação dos padrões de evolução identificados, visando reduzir os custos e os problemas futuros;
- criar um arcabouço e um conjunto de operações que permitam quantificar o custo associado aos vários padrões de evolução, possibilitando a análise do retorno sobre o investimento;
- analisar a família de padrões de evolução identificada no ambiente de desenvolvimento de outras empresas, que possuem múltiplas linhas de produto em paralelo.

Com relação ao processo para a escolha do padrão de evolução mais adequado ao se implementar uma determinada requisição de mudança, pode-se:

- analisar a utilização do processo no ambiente de desenvolvimento de outras empresas, que possuem múltiplas linhas de produto em paralelo.
- estender o processo de forma a criar um único processo de análise para evolução de componentes em linhas de produto.
- estender o processo para apoiar requisições de mudanças simultâneas, ou seja, vindo de diversas linhas de produtos e que afetam mais de uma característica implementada pelo componente.
- disponibilizar apoio computacional para o processo.

Referências Bibliográficas

[1] Colin Atkinson, Joachim Bayer, and Dirk Muthig. Component based product line development – the kobra approach. Technical report, Fraunhofer Institute for Experimental Software Engineering, 1999.

[2] Felix Bachmann and Len Bass. Managing variability in software architectures. *SIGSOFT Softw. Eng. Notes*, 26(3):126–132, 2001.

[3] Stefan Van Baelen, David Urting, Werner Van Belle, Viviane Jonckers, Tom Holvoet, Yolande Berbers, and Karel De Vlaminck. Toward a unified terminology for component-based development. In *14th European Conference on Object-Oriented Programming (ECOOP)*, 2000.

[4] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. Addison Wesley, 2 edition, 2003.

[5] Joachim Bayer, Oliver Flege, and Cristina Gacek. Creating product line architectures. *Lecture Notes in Computer Science Proceedings of the International Workshop on Software Architectures for Product Families*, 1951:210–216, 2000.

[6] Keith H. Bennett and Václav T. Rajlich. Software maintenance and evolution: a roadmap. *International Conference on Software Engineering*, 2000.

[7] Danilo Beuche. Transforming legacy systems into software product lines. In *Proceedings of the 10th International on Software Product Line Conference*. IEEE Computer Society, 2006.

[8] Jan Bosch. Evolution and composition of reusable assets in product-line architectures: A case study. *First Working IFIP Conference on Software Architecture*, 1999.

[9] Jan Bosch and PO Bengtsson. Component evolution in product line architectures. In *Proceedings of the International Workshop on Component-Based Software Engineering*, 1999.

[10] Paul Clements and Linda Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley, 2002.

[11] Reidar Conradi and Bernhard Westfechtel. Version models for software configuration management. *ACM Computing Surveys*, 30:232–282, 1998.

[12] Krzysztof Czarnecki and Ulrich Eisenecker. *Generative Programming: Methods, Tools and Applications*. Addison-Wesley, 2000.

[13] Krzysztof Czarnecki, Simon Helsen, and Ulrich Eisenecker. Formalizing cardinality-based feature models and their specialization. *Software Process: Improvement and Practice*, 10:7–29, 2005.

[14] Moacir Caetano da Silva Júnior. Cosmos - um modelo de estruturacão de componentes para sistemas orientados a objetos. Master's thesis, Universidade Estadual de Campinas (Unicamp), 2003.

[15] Eduardo Figueiredo, Nelio Cacho, Claudio Sant'Anna, Mario Monteiro, Uira Kulesza, Alessandro Garcia, Sergio Soares, Fabiano Ferrari, Safoora Khan1, Fernando Castor Filho, and Francisco Dantas. Evolving software product lines with aspects: An empirical study on design stability. *International Conference on Software Engineering*, 2008.

[16] Erich Gamma, Richard Helm, Ralph Johnson, and John M. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994.

[17] Leonel Aguilar Gayard, Cecília Mary Fischer Rubira, and Paulo Asterio de Castro Guerra. Cosmos*: a component system model for software architectures. Technical report, Instituto de Computação, Universidade Estadual de Campinas, 2008.

[18] David Gibson, Bruce W. Weide, Scott Pike, and Stephen Edwards. Toward a normative theory for component-based system design and analysis. *Cambridge University Press*, pages 211 – 230, 200.

[19] Jilles Van Gurp, Jan Bosch, and Mikael Svahnberg. On the notion of variability in software product lines. In IEEE Computer Society, editor, *Working IEEE/IFIP Conference on Software Architecture (WICSA'01)*, page 45, 2001.

[20] Kyo Kang, Sholom Cohen, James Hess, William Novak, and Spencer Peterson. Feature-oriented domain analysis (foda) feasibility study. Technical report, Carnegie Mellon University - Software Engineering Institute, 1990.

[21] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-oriented programming. *European Conference on Object-Oriented Programming (ECOOP), Finland. Springer-Verlag LNCS 1241*, 1997.

[22] Charles Krueger. Variation management for software production lines. In *2nd International Software Product Line Conference, volume 2379 of LNCS, pages 37.48*, 2002.

[23] Manny Lehman. Laws of software evolution revisited. *Lecture Notes in Computer Science*, 1149/1996:108–124, 1996.

[24] John D. McGregor. The evolution of product line assets. Technical report, Software Engineering Institute, 2003.

[25] Kannan Mohan and Balasubramaniam Ramesh. Change management patterns in software product lines. *Communications of the ACM*, 49, 2006.

[26] Linda Northrop. Software product line adoption roadmap. Technical report, Carnegie Mellon University - Software Engineering Institute, 2004.

[27] Linda Northrop. A framework for software product line practice, version 4.2, Agosto 2007.

[28] Claudio Riva and Christian Del Rosso. Experiences with software product family evolution. In *6th International Workshop on Principles of Software Evolution*, 2003.

[29] IEEE Computer Society and Association for Computing Machinery. Software engineering 2004: Curriculum guidelines for undergraduate degree programs in software engineering, Agosto 2004.

[30] Ian Sommerville. *Software Engineering*. Addison-Wesley, 8 edition, 2006.

[31] Alexander Stuckenholz. Component evolution and versioning state of the art. *ACM SIGSOFT Software Engineering Notes*, 30, 2005.

[32] Mikael Svahnberg and Jan Bosch. Characterizing evolution in product line architectures. In *3rd annual IASTED International Conference on Software Engineering and Applications*, 1999.

[33] Clemens Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley, 2002.

[34] Louis J.M. Taborda. Planning and managing product line evolution. *Proceedings of Fifth Product Family Engineering (PFE-5), LNCS*, 2003.

[35] Louis J.M. Taborda. The release matrix for component-based software systems. *Lecture Notes in Computer Science*, 3054/2004:100–113, 2004.

[36] Amir Tomer, Leah Goldin, Tsvi Kuflik, Esther Kimchi, and Stephen R. Schach. Evaluating software reuse alternatives: A model and its application to an industrial case study. *IEEE Transactions on Software Engineering*, 30:601–612, 2004.

[37] Rob van Ommering. Configuration management in component based product populations. In *Tenth International Workshop on Software Configuration Management*, 2001.

[38] Trevor J. Young. Using aspectj to build a software product line for mobile devices. Master's thesis, University of Waterloo, 2005.