Universidade Federal de São Carlos

Centro de Ciências Exatas e de Tecnologia

Programa de Pós-Graduação em Ciência da Computação

"Uma Abordagem de Reengenharia Iterativa Orientada a Características para Sistemas Embutidos Legados"

MARCELO AUGUSTO RAMOS

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Membros da Banca:

Profa. Dra. Rosângela Ap. Dellosso Penteado
(Orientadora – DC/UFSCar)

Prof. Dr. Paulo César Masiero
(ICMC/USP)

Profa. Dra. Claudia Maria Lima Werner
(COPPE/UFRJ)

São Carlos
Agosto/2007

Dedico este trabalho a DEUS, por me dar saúde, paz, prosperidade e felicidade;

Ao meu pai Waster U. Ramos (*in memoriam*), por me ensinar a não desistir dos meus sonhos;

À minha mãe Zeneide B. Ramos (*in memoriam*), por dedicar sua vida ao meu bem estar;

À minha tia Wamir S. Ramos, por me ter como a um filho;

À minha esposa Adriana G. A. Ramos, por me amar incondicionalmente;

Aos meus filhos Juliana e Gabriel, por sua compreensão e carinho.

Agradecimentos

Agradeço primeiramente à minha orientadora, Profª Drª Rosângela A. D. Penteado, por me estender a mão nos momentos mais difíceis desta caminhada, pelas horas dedicadas à correção de meus trabalhos, pelos incontáveis e-mails que atenciosamente respondeu e por compartilhar comigo toda a sua experiência profissional;

Aos Profes Dres Antonio F. do Prado e Mauro Biajiz que aprovaram minha admissão no programa de mestrado;

E, por fim, agradeço a todas as pessoas que direta ou indiretamente contribuíram para esta conquista tão importante.

A todos vocês, meu sincero MUITO OBRIGADO!!

Sumário

Lista de Figuras

Lista de Tabelas

Resumo

Para reduzir custos, minimizar riscos, antecipar prazos e otimizar recursos de projetos de novos produtos recomenda-se, sempre que possível, a reutilização de artefatos de produtos similares e bem sucedidos existentes. Porém, esses artefatos devem ser de fácil adaptação para que satisfaçam aos requisitos desses novos produtos com pouca ou nenhuma necessidade de re-projeto ou de re-codificação. Este trabalho descreve uma abordagem de reengenharia iterativa orientada a características para realizar concomitantemente a revitalização de sistemas embutidos legados e a criação incremental de um núcleo de artefatos reutilizáveis, para apoiar o desenvolvimento de outros sistemas similares, membros de uma família de produtos. Tais sistemas são geralmente pequenos e sofrem constantes manutenções para apoiarem a evolução do hardware que integram, acelerando o processo natural de degradação do software. A realização de um processo de reengenharia pode prolongar a vida útil desses sistemas, melhorando suas propriedades estruturais para uma melhor adaptação a mudanças. Princípios e técnicas ágeis são aplicados ao longo de todo o processo, provendo interações contínuas com o cliente, entregas de versões executáveis testadas e parcialmente modernizadas em intervalos curtos e freqüentes, com documentação em nível apropriado e preparadas para modificações futuras. Técnicas de Linha de Produtos de Software são utilizadas para a modelagem de domínio e para o projeto de componentes genéricos de software. Suas atividades são realizadas sem a necessidade de congelamentos ou duplicações de código e permitem que manutenções e paralisações ocorram a qualquer instante, sem prejuízo das melhorias já desenvolvidas nas iterações previamente concluídas. Um estudo de caso é apresentado para exemplificar a aplicação da abordagem proposta em um ambiente real e para avaliar os seus resultados.

Abstract

To reduce costs, to minimize risks, to anticipate deadlines and to optimize resources of new product designs it is recommended, whenever possible, to reuse artifacts of existing successful similar products. However, these artifacts must be of easy adaptation to meet the requirements of the new products with a few or no need of re-design or re-coding. This work describes a feature oriented iterative reengineering approach to concomitantly achieve both the revitalization of embedded legacy systems and the incremental creation of a core of reusable artifacts, to support the development of other similar systems, members of a product family. Such systems are generally small and pass through frequent maintenances to support the evolution of the hardware they integrate, accelerating the natural software degradation process. The accomplishment of a reengineering process can extend the life cycle of these systems, improving their structural properties for a better adaptation to changes. Agile principles and techniques are applied throughout the process, providing continuous interactions with the customer and deliveries of executable and tested partially modernized versions in short and frequent intervals, with documentation in appropriate level and prepared for future modifications. Software Product Line techniques are used for the domain modeling and for the design of generic software components. Its activities are accomplished without the need of code freezing or duplication and allow maintenances and interruptions at anytime, without prejudice of the enhancements already developed in all previously ended iterations. A case study is presented to exemplify the proposed approach in a real environment and to evaluate its results.

Capítulo 1

Introdução

1.1 Considerações Iniciais

Um sistema embutido do ponto de vista de seus dispositivos físicos vizinhos é apenas outro dispositivo físico com características físicas, como acionadores mecânicos etc. O software embutido nesses equipamentos provê uma maneira de customizar muitas de suas características físicas programáveis. Muitos produtos hoje em dia possuem software embutido, como por exemplo, equipamentos médicos, aparelhos de DVD etc. (VAHID; GIVARGIS, 2002). Devido aos avanços da tecnologia da informação e da comunicação, cada vez mais e mais produtos irão possuí-los. Essa crescente demanda tem aumentado a sua complexidade, quantidade e variedade, criando um grande desafio para o seu desenvolvimento. O sucesso contínuo dessa tarefa está associado, entre outros fatores, à capacidade de desenvolvê-los com qualidade, rapidez e baixo custo. Porém, muitas empresas podem ter problemas em alcançar esses objetivos diante da necessidade constante de evolução de seus produtos. Para melhorar a produtividade e a qualidade de seus sistemas, as empresas buscam aplicar tecnologias da engenharia de software apropriadas para cada situação específica. No entanto, muitas dessas tecnologias não consideram necessidades específicas associadas ao desenvolvimento de software embutido, como por exemplo, variações do hardware, memória limitada e consumo de energia. Adaptar tecnologias para atender necessidades específicas pode ser uma tarefa difícil. Adicionalmente, o domínio de software embutido é guiado por fatores de confiabilidade, de custo e de tempo, necessitando assim, de tecnologias de desenvolvimento voltadas para a produtividade. Apesar do sentimento existente em muitas empresas de que seus processos atuais não são plenamente satisfatórios, mudanças devem ocorrer de forma gradual. Para isso, é necessário um maior conhecimento dos métodos, ferramentas e técnicas atualmente em uso nas empresas para que se possa oferecer uma proposta realista de evolução (GRAAF; LORMANS; TOETENEL, 2003).

Para alcançar a produtividade necessária, muitas equipes de desenvolvimento de software embutido procuram definir uma metodologia de projeto própria, que agregue um

diferencial de seus concorrentes. Estas metodologias em geral baseiam-se nos recursos disponíveis, utilizando partes ou experiências de projetos anteriores. Porém, muitas vezes não é dada uma atenção especial à atualização da documentação, dificultando progressivamente o desenvolvimento de novos produtos baseado nessa forma de reúso (SZTIPANOVITS, 2000).

O reúso é uma solução parcial para o problema de produtividade no desenvolvimento de software embutido, que evolui e muda ao longo do tempo. Muitas empresas possuem bibliotecas de módulos de software reutilizáveis, muitos dos quais são usados em múltiplas aplicações. Nesses casos, muitos dos relatos de sucesso têm sido prematuros. Geralmente, esses módulos podem ser reutilizados em projetos similares próximos, mas poucos realmente sobrevivem ao tempo e permanecem sendo reutilizados em projetos futuros. Esse fato não exclui o reúso de software como solução para melhoria da produtividade, mas sugere alguns requisitos para a sua prática, que inclui a possibilidade de realizar manutenções nas aplicações de maneira fácil, rápida e segura (LEVESON; WEISS, 2004).

Desenvolvimento Baseado em Componentes (DBC) (HEINEMAN; COUNCILL, 2001) é uma técnica de desenvolvimento voltada para o reúso, que tem despertado o interesse de muitas empresas. Porém, a experiência mostra que é difícil usar uma mesma tecnologia de componentes de software para diferentes domínios, devido aos seus diferentes requisitos e limitações. No domínio de sistemas embutidos, por exemplo, componentes, geralmente, não são elementos executáveis separados da aplicação, mas sim códigos escritos em linguagens de alto nível, que se unem à aplicação durante a sua criação (*build-time*). Outro fato importante é que muitas das tecnologias modernas ainda não apóiam requisitos indispensáveis para o desenvolvimento de componentes de software para sistemas embutidos, como o consumo reduzido de memória e a performance (CRNKOVIC, 2005).

Linha de Produtos de Software (LPS) (CLEMENTS; NORTHROP, 2001) é uma técnica de desenvolvimento que reutiliza artefatos, *i.e.*, requisitos, arquitetura e componentes, em um nível mais elevado de abstração. A idéia principal é desenvolver uma infra-estrutura reutilizável que apóie o desenvolvimento de software para uma família de produtos similares. Para determinar quando essa técnica é aplicável, uma empresa deve avaliar o nível de generalidade de seus produtos, o conhecimento do domínio de seus profissionais e a qualidade dos artefatos de software existentes (NIEMELÄ; IHME, 2001).

No Capítulo 2 é feita uma análise mais detalhada dessas e de outras técnicas de desenvolvimento de software, que possibilitam melhorar a produtividade do desenvolvimento de software embutido.

1.2 Objetivo

O objetivo deste trabalho é propor uma abordagem de reengenharia iterativa que reúne técnicas e tecnologias da engenharia de software e as utiliza, de forma integral ou adaptada, para realizar concomitantemente a revitalização de sistemas embutidos legados e a criação incremental de um núcleo de artefatos reutilizáveis. Esses artefatos devem apoiar o desenvolvimento de outros sistemas similares, membros de uma família de produtos. Espera-se com a realização da reengenharia prolongar a vida útil de sistemas embutidos legados, agregando-lhes o apoio a novas variabilidades do domínio, por meio do reúso total ou parcial de artefatos existentes, permitindo que novos produtos similares possam ser criados rapidamente, aumentado a produtividade da empresa. Deve-se melhorar gradualmente a forma de reúso atual, permitindo uma adaptação mais fácil da equipe de desenvolvimento em relação ao uso de técnicas mais modernas da engenharia de software. Busca-se também com essa abordagem realizar as tarefas mencionadas com baixo risco e custo para a empresa, favorecendo a sua aceitação na prática.

1.3 Motivação

A longa experiência profissional do autor desta dissertação em desenvolvimento de software embutido para o domínio de aplicações para TEF (Transferência Eletrônica de Fundos) com cartões de pagamento, embutidas em terminais POS (*Point Of Sale*), permite atestar a grande variedade de hardware e plataformas de software existentes (VERIFONE BR). A agregação contínua de novos serviços às aplicações em uso, decorrente das oportunidades de negócio criadas pelo mercado consumidor, aliada à concorrência entre empresas fornecedoras de soluções para o domínio, impulsionam a evolução contínua do hardware e exigem freqüentes manutenções do software, em prazos cada vez menores, acelerando a sua degradação. Devido à existência de informações corporativas confidenciais do cliente e do fornecedor, protegidas por contratos de sigilo, os ambientes de

desenvolvimento dessas empresas são comumente fechados e integram soluções particulares, porém não necessariamente modernas, para esse problema.

Este trabalho conta com o apoio de uma empresa real atuante nesse domínio, cujo objetivo é aplicar modernas técnicas e tecnologias da engenharia de software para revitalizar suas aplicações para TEF com cartões de pagamento, embutidas em terminais POS, e, a partir delas, ser capaz de desenvolver rapidamente novos produtos similares. Assim sendo, o conhecimento do domínio e a possibilidade de acesso à infra-estrutura de desenvolvimento de uma empresa real para a realização desse trabalho constituem os principais fatores de motivação.

1.4 Organização da Dissertação

Esta dissertação está organizada da seguinte forma: o Capítulo 2 apresenta conceitos, tecnologias e técnicas presentes na literatura, relacionados ao contexto deste trabalho e relevantes para a sua realização; o Capítulo 3 descreve a abordagem de reengenharia iterativa orientada a características proposta, com suas fases, atividades e artefatos utilizados e produzidos; o Capítulo 4 mostra um estudo de caso realizado no domínio de terminais POS para exemplificar a aplicação da abordagem proposta em ambiente real e avaliar os seus resultados e, por fim, no Capítulo 5 são apresentadas as conclusões e os trabalhos futuros.

Capítulo 2

Conceitos e Trabalhos Relacionados

2.1 Considerações Iniciais

O aperfeiçoamento constante dos processos automatizados de produção industrial tem ocasionado uma redução progressiva do tempo entre a concepção de um novo produto manufaturado e a sua oferta ao mercado consumidor[1]. Uma das estratégias para aumentar a oferta desse produto, atendendo às diferentes necessidades do mercado, consiste em projetá-lo de forma genérica, tornando-o capaz de apoiar a rápida criação de diferentes modelos derivados, por meio da modificação planejada de seus elementos mecânicos e eletrônicos. Se necessário garantir-lhe uma sobrevida, seja para ampliar as oportunidades de negócio ou para maximizar o retorno dos investimentos realizados, novas variações, inicialmente não planejadas, podem ser criadas ao longo do seu ciclo de vida. Quando parte da funcionalidade desse produto é provida por software, esse deve ser capaz de acompanhar agilmente todo o processo de evolução do produto no qual se encontra inserido. Para isso, pode ser necessária a realização de manutenções freqüentes, que podem ser apoiadas por técnicas de engenharia de software para preservar a manutenibilidade do código e maximizar o reúso de soluções já desenvolvidas para modelos anteriores do produto.

Manutenção pode ser vista como um caso particular de desenvolvimento de software orientado para o reúso em um modelo de processo no qual a idéia central é a criação de um repositório comum de artefatos para sistemas de um mesmo domínio. Dessa forma, sua realização propicia a obtenção de um núcleo de artefatos reutilizáveis, em diferentes níveis de abstração, facilitando a implementação de novos sistemas, com menor custo e esforço (BASILI, 1990). A reengenharia de sistemas, classificada como manutenção preventiva (PRESSMAN, 2006), pode ser planejada com base nessa proposta e incluir atividades voltadas para a criação de artefatos reutilizáveis, que possam compor a base para o desenvolvimento ágil de novos produtos. Uma prática realista pode iniciar essa tarefa analisando sistemas similares de um domínio, para entender sua arquitetura atual, e

[1] *Time-to-Market*

desenvolvendo uma estratégia para mineração e reutilização dos recursos comuns existentes (SEI).

Este capítulo reúne conceitos essenciais que apóiam a abordagem de reengenharia proposta e inclui trabalhos relacionados, presentes na literatura, que integram seus fundamentos. A abordagem proposta utiliza os conceitos e os resultados dos trabalhos aqui relacionados, de forma integral ou adaptada, para realizar a reengenharia de sistemas embutidos legados, promovendo sua revitalização, concomitantemente com a criação de um núcleo de artefatos reutilizáveis, para apoio ao desenvolvimento ágil de produtos similares, membros de uma família de produtos de software.

As demais seções estão organizadas da seguinte forma: a Seção 2.2 contextualiza a existência de software legado no domínio de sistemas embutidos e cita problemas associados à sua manutenção; a Seção 2.3 descreve o conceito de refatoração, uma técnica para a melhoria da integridade estrutural de códigos legados; a Seção 2.4 descreve princípios genéricos que moldam um processo ágil de desenvolvimento e relata uma das possíveis causas de insucesso na sua aplicação; a Seção 2.5 apresenta reúso de software e seus benefícios para a criação de novos produtos; a Seção 2.6 conceitua componentes de software, descreve sua importância para o reúso e apresenta técnicas para mineração de componentes a partir de software legado; a Seção 2.7 apresenta Linha de Produtos de Software (LPS), um paradigma de desenvolvimento voltado para o reúso em níveis de abstração mais altos, e trabalhos existentes na literatura que fazem uso de seus conceitos e técnicas; a Seção 2.8 conceitua manutenção e reengenharia de software e apresenta formas de realizá-las visando à criação de artefatos reutilizáveis e, finalmente, na Seção 2.9 são feitas as considerações finais do capítulo.

2.2 Sistemas Embutidos e Software Legado

Diferentes de um computador pessoal de propósito geral, sistemas embutidos são sistemas computacionais especializados, hardware e software, que integram sistemas com funcionalidade mais ampla, realizando tarefas específicas e pré-definidas como criptografias, controle de temperatura, etc. Sua constituição pode variar de um único chip FPGA[2] a um circuito com microprocessador e sistema operacional. Software escrito para sistemas

[2] Chip programável, com código de programa fixo gravado internamente.

embutidos é freqüentemente chamado de software embutido ou *firmware*, e é armazenado em chips de memória ROM[3] ou Flash[4]. Apesar da variedade de projetos existentes, esse software freqüentemente opera em hardware com um conjunto reduzido de entradas e saídas e capacidade limitada de armazenamento e processamento de informações (VAHID; GIVARGIS, 2002).

Com o passar do tempo um sistema de software pode sofrer modificações para suprir suas necessidades naturais de evolução como correção de falhas, acomodação de modificações e implementação de novos requisitos. Seu grau de importância para os negócios da empresa usuária pode determinar seu uso prolongado, mesmo diante da obsolescência de sua tecnologia e da degradação de seus artefatos, passando a integrar a base de software legado dessa empresa (SEACORD; PLAKOSH; LEWIS, 2003). O código desses sistemas pode ser difícil de ser compreendido, por vários motivos: criação por método *ad hoc*, falta de documentação de alto nível, falta de integridade conceitual de sua arquitetura e projeto, entre outros.

Apenas 2% dos processadores produzidos atualmente são empregados em computadores propriamente ditos como PC's etc. Os 98% restantes equipam sistemas embutidos (TENNENHOUSE, 2000). O uso em larga escala desses sistemas impulsiona a contínua diversificação e evolução do hardware, exigindo constantes manutenções do software embutido, acelerando a sua degradação. O número de soluções disponíveis para minimizar esse efeito é reduzido, pois tecnologias modernas de engenharia de software geralmente não consideram necessidades específicas associadas ao desenvolvimento de software embutido e restrições usuais desse domínio, como a limitação de memória, o consumo de energia e as variações do hardware (GRAAF; LORMANS; TOETENEL, 2003).

2.3 Refatoração

Quando uma quantidade significativa de códigos legados e pobremente projetados passou a compor a base de aplicações ineficientes e difíceis de manter e estender, um conjunto crescente de técnicas, denominadas "refatoração" (FOWLER et al., 2004), começou a ser desenvolvido e ser utilizado por programadores para melhorar a integridade estrutural e

[3] Memória apenas de leitura (do termo em inglês *Read Only Memory*).
[4] Memória não volátil que pode ser eletricamente apagada.

o desempenho de sistemas de software existentes, sem prejuízo de suas funcionalidades. Fowler et al. (2004) ressaltam a existência de duas definições distintas:

Refatoração (Substantivo): alteração feita na estrutura interna de um software para torná-lo mais fácil de ser entendido e menos trabalhoso de ser modificado, sem alterar seu comportamento observável;

Refatorar (Verbo): reestruturar o software aplicando uma série de refatorações.

Seu trabalho demonstra como desenvolvedores de software podem tornar reais os benefícios significativos dessa técnica. Mostra ainda que, com treinamento apropriado, um projetista de sistemas pode, a partir de um código ruim, obter um código robusto e bem projetado. Esse processo, no entanto, não é livre de falhas e deve ser executado gradativamente com o apoio de um conjunto sólido de testes para garantir a preservação da funcionalidade observável.

2.4 Princípios Ágeis

Princípios ágeis moldam um processo de desenvolvimento de software caracterizado por um forte espírito de equipe (cooperativo), simplicidade e velocidade, com entregas de versões parciais executáveis e testadas (incremental), em intervalos curtos e freqüentes (iterativo), com documentação em nível apropriado (direto), envolvimento do cliente no processo (interativo) e preparação para mudanças (adaptativo) (ABRAHAMSSON et al., 2002).

Metodologias ágeis enfocam o desenvolvimento rápido de software, no qual programadores, em contato com o cliente, desenvolvem múltiplas versões de um sistema, até que tenham produzido um software funcional que o satisfaça. Apesar dos relatos de sucesso, Gotterbarn (2004) alerta para o problema de que sistemas estão sendo desenvolvidos como se fossem isolados, sendo observados somente os pontos de vista de seus programadores e do próprio cliente, uma vez que essas metodologias não vêem o software como parte de um sistema funcional mais amplo, que interage com uma variedade de pessoas. Segundo ele, essa tem sido a causa de muitas falhas.

2.5 Reúso de Software

Reúso de software se refere ao uso de artefatos existentes para apoiar a construção de novos sistemas, ao invés de desenvolvê-los integralmente a partir do zero (KRUEGER, 2002). Essa prática favorece a criação de sistemas de boa qualidade, com custos reduzidos e maior flexibilidade, permitindo modificações futuras com menor esforço (JACOBSON; GRISS; JONSSON, 1997).

Bosch (2000) apresenta duas formas de classificar o reúso de software, i.e., oportunista ou planejado e *bottom-up* ou *top-down*.

- Oportunista: Pedaços de software, relacionados a um problema específico, são selecionados, combinados e adicionados a um produto em desenvolvimento.
- Planejado: Artefatos reutilizáveis são produzidos mediante esforço para prover abstrações e níveis de variabilidades adequados para um produto planejado.
- *bottom-up*: Componentes reutilizáveis, uma vez desenvolvidos ou minerados, são adicionados a uma coleção, possivelmente grande, de artefatos de software, da qual podem ser extraídos para a solução de problemas recorrentes.
- *top-down*: Artefatos de software são produzidos como partes integrantes de uma estrutura de mais alto nível, unido-se a interfaces providas e requeridas pré-definidas.

Bosch (2000) afirma, com base em relatos da comunidade de reúso, que reúso oportunista e *bottom-up* não são bem sucedidos na prática.

Novas técnicas têm sido pesquisadas com a finalidade de elevar o reúso a níveis de abstração mais altos, como por exemplo, Componentes (HEINEMAN; COUNCILL, 2001) e Linha de Produtos de Software (CLEMENTS; NORTHROP, 2001). Técnicas de reúso com altos níveis de abstração facilitam o desenvolvimento de sistemas semelhantes, mediante o reaproveitamento das similaridades e o gerenciamento das variabilidades neles existentes (KANG et al., 1990). Krueger (2002) apresenta uma metodologia com apoio ferramental para massificação de software por meio de customizações. Uma de suas técnicas, denominada extrativa, possibilita rápida transformação de sistemas legados em artefatos de uma linha de produção, que encapsula similaridades e variabilidades parametrizáveis desses sistemas na forma de características. A criação de novos produtos consiste no uso de ferramentas para agrupar as características desejadas, fornecendo parâmetros para as variabilidades selecionadas.

2.6 Componentes de Software e Mineração

Componente é uma unidade de software independente, que encapsula, dentro de si, seu projeto de implementação e oferece serviços, por meio de interfaces bem definidas, para o meio externo (BARROCA; GIMENES; HUZITA, 2005). Componentes com alto grau de generalidade e fácil adaptação podem prover soluções reutilizáveis para requisitos similares em diferentes sistemas. Para isso, devem agregar propriedades intrínsecas dos domínios aos quais se aplicam, para apoiar a implementação das variabilidades neles existentes (BERGEY; O'BRIEN; SMITH, 2000).

Mineração frequentemente se refere ao processo de extração não trivial de artefatos de software potencialmente úteis e reutilizáveis, embutidos nos sistemas legados de uma empresa (BERGEY; O'BRIEN; SMITH, 2000).

Mehta e Heineman (2002) propõem uma metodologia para a modernização de sistemas de software, que agrega características, testes de regressão e engenharia de software baseada em componentes. A proposta prevê a modernização de um conjunto de características identificadas durante testes de regressão. O código associado a cada característica é identificado, refatorado, convertido em componente de software e inserido novamente na aplicação. Os componentes criados podem ser reutilizados em outras aplicações. O'Brien (2005) observa que se os paradigmas de desenvolvimento do sistema legado e dos componentes recém criados forem distintos, *gateways* devem ser desenvolvidos para permitir a re-conexão de seus códigos.

2.7 Linha de Produtos de Software

Linha de Produtos de Software (LPS) consiste em um conjunto, ou família, de produtos pertencentes a um determinado domínio, que tem como base uma arquitetura comum. Seu objetivo é ampliar a eficiência dos processos de desenvolvimento, explorando a identificação e o reúso das similaridades e controlando as variabilidades dos membros de uma família. Com base nesse princípio, projetistas de software atuam na elaboração não mais de um simples produto, que possa vir a ser alterado, mas de um produto genérico, que favoreça a rápida produção de diferentes versões, ou membros da família de produtos, adequadas às necessidades do mercado. Novos membros são criados com pouca ou nenhuma necessidade de revisão ou re-codificação do projeto (CLEMENTS; NORTHROP, 2001).

Um processo genérico de engenharia para LPS é constituído de duas atividades principais, *i.e.*, a Engenharia de Domínio e a Engenharia de Aplicação, como mostra a Figura 1 (ZIADI; JÉZÉQUEL; FONDEMENT, 2003).

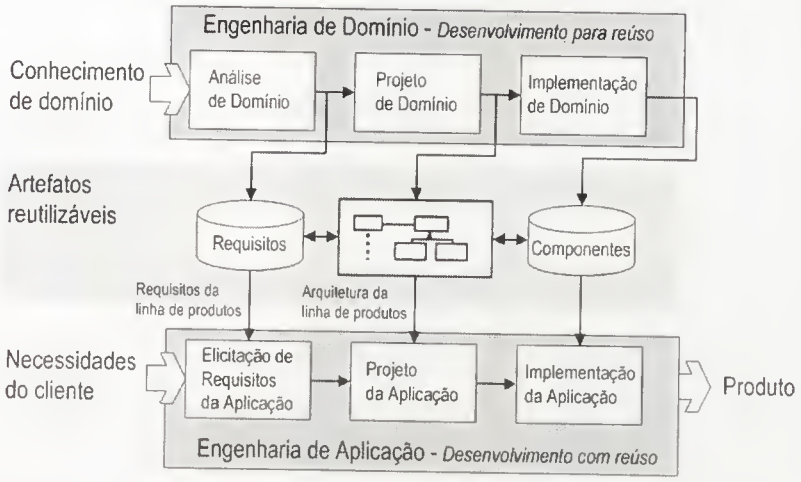


Figura 1. Processo genérico de engenharia para LPS [Adaptado de (ZIADI; JÉZÉQUEL; FONDEMENT, 2003)]

A atividade de Engenharia de Domínio, ou Desenvolvimento para reúso, pode ser dividida em três fases principais: Análise, Projeto e Implementação de Domínio. A Análise de Domínio faz a elicitação dos requisitos da LPS. O Projeto de Domínio os utiliza para criar a arquitetura da LPS, que a Implementação de Domínio usa para desenvolver os componentes de software. A atividade de Engenharia de Aplicação, ou Desenvolvimento com reúso, cria novos produtos a partir dos artefatos produzidos pela Engenharia de Domínio, que formam o núcleo de ativos[5] da LPS, composto por seus requisitos, modelos de domínio, arquiteturas, componentes de software, etc.

Com base nesse processo, vários métodos de desenvolvimento para LPS foram propostos como FODA/FORM (KANG et al., 1990, 1998), FAST (WEISS; LAI, 1999), KobrA (ATKINSON et al., 2001) entre outros.

A Figura 2 mostra resumidamente as atividades do método FAST (Family-Oriented Abstraction Specification and Translation) (WEISS; LAI, 1999).

[5] Tradução do termo original em inglês *core assets*

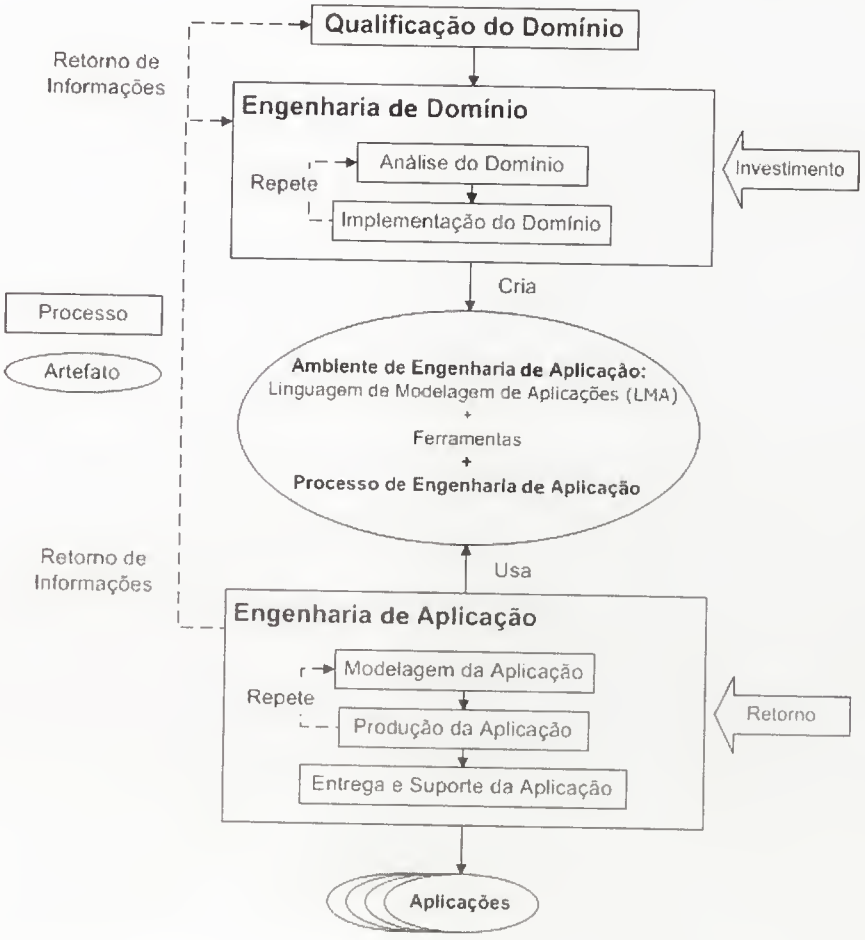


Figura 2. Atividades do método FAST [Adaptado de (WEISS; LAI, 1999)]

A Qualificação do Domínio analisa a viabilidade econômica de uma família de produtos, estimando a quantidade e o valor de seus membros e seus respectivos custos de implementação. A Engenharia de Domínio define a família e provê meios para a implementação de seus membros, criando um Ambiente de Engenharia de Aplicação que inclui: a) uma LMA (**L**inguagem de **M**odelagem de **A**plicações), que apóia a modelagem dos membros da família e b) uma ferramenta, que apóia a produção desses a partir de seus requisitos. Para Weiss e Lai (1999), essa é a etapa de maior investimento e riscos do processo. A Engenharia de Aplicação é um processo iterativo que faz uso intenso do ambiente criado pela Engenharia de Domínio para criar os membros da família por meio das seguintes atividades: análise e validação dos requisitos do cliente para um membro da família, modelagem do membro da família com a LMA, produção do membro da família (código e documentação), entrega da aplicação e documentação ao cliente e suporte operacional da

aplicação. Para Weiss e Lai (1999), devido à rápida produção dos membros da família, esta é a etapa de retorno do investimento.

KobrA (ATKINSON et al., 2001) é um método de desenvolvimento apoiado por três atividades principais, Decomposição, Instanciação e Concretização de artefatos, que lidam distintamente com seus níveis de composição, generalização e abstração, organizados na forma de três dimensões ortogonais, ilustradas na Figura 3.

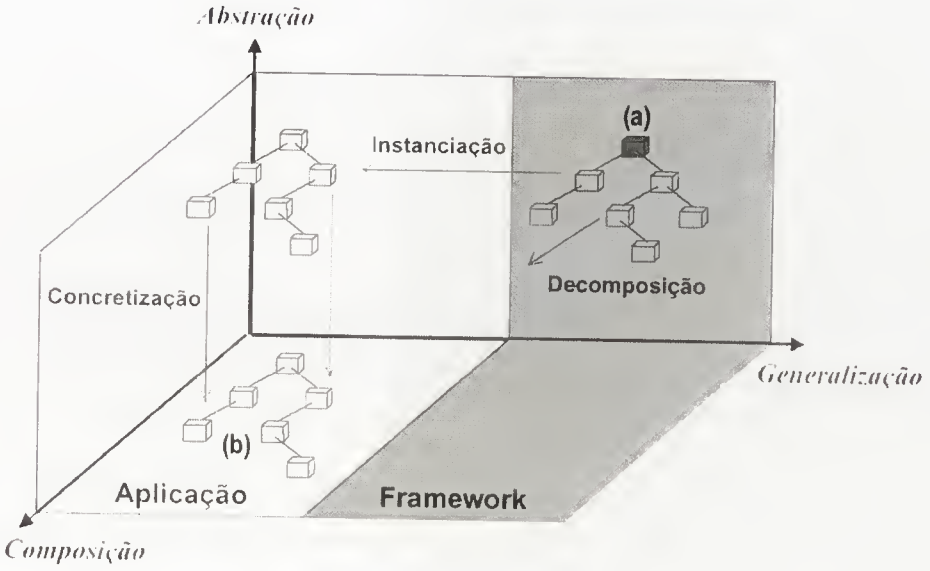


Figura 3. Atividades do método [Adaptado de (ATKINSON et al., 2001)]

Um típico projeto KobrA começa com a descrição do sistema a ser desenvolvido na forma de uma caixa preta genérica e abstrata, Figura 3(a). É caixa preta porque descreve apenas as propriedades externamente visíveis do sistema, como um componente único, genérica porque provavelmente possui características variantes (parametrizáveis) e abstrata porque é formada por modelos UML (BOOCH; RUMBAUGH; JACOBSON, 1999) em nível similar ao de análise e de projeto. Para a criação de uma versão executável do sistema, é necessário remover a generalidade, a fim de criar uma instância do sistema que satisfaça às necessidades de uma aplicação específica (instanciação), decompor o sistema em pequenas partes para produzir uma árvore ramificada de componentes aninhados (decomposição) e reduzir o nível de abstração para criar uma representação do sistema e suas partes (concretização). A realização dessas três transformações produz uma aplicação, Figura 3(b),

representada em termos de componentes detalhados, com baixa generalidade (i.e. zero) e uma representação completa (i.e. um executável). Para o desenvolvimento de uma LPS, o enfoque é direcionado para a generalidade e o ciclo de vida completo do software consiste em duas atividades: Engenharia de *Framework* e Engenharia de Aplicação, Figura 4.

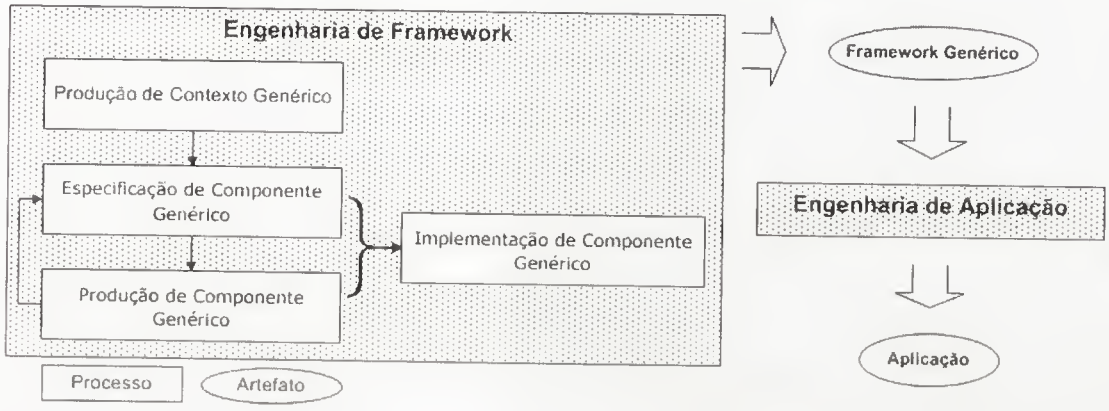


Figura 4. Ciclo de vida de uma aplicação KobrA com foco em LPS [Adaptado de (ATKINSON et al., 2001)]

Quando aplicada de forma genérica na Engenharia de *Framework*, a Produção de Contexto corresponde à tarefa tradicional de Análise de Domínio da Engenharia de Domínio e o processo recursivo de Especificação e Produção de Componentes Genéricos corresponde à construção da arquitetura de referência e dos componentes arquiteturais para um domínio específico. Durante a Engenharia de Aplicação o *Framework* Genérico criado pela Engenharia de *Framework* é instanciado para a construção de aplicações particulares. Essa fase também inclui atividades de modelagem e implementação, mas aplicadas de modo específico para criação de produtos (componentes) específicos.

FODA (*Feature Oriented Domain Analysis*) (KANG et al., 1990) é um método de análise de domínio, que dá ênfase à identificação de características importantes ou especiais de um domínio, visíveis ao usuário e embutidas em uma classe de sistemas de software relacionados. Esses sistemas são abstraídos até um nível em que não haja diferenças entre eles, resultando na criação de produtos genéricos. Esses produtos, por sua vez, são refinados de formas distintas para a criação de membros de uma família de produtos. Dessa forma, cada produto é composto por uma coleção de abstrações e por uma série de refinamentos de cada abstração, realizados por meio de parâmetros.

Diferentemente das abordagens FAST (WEISS; LAI, 1999) e KobrA (ATKINSON et al., 2001), a análise de domínio no método FODA é baseada na construção de modelos de características. Neles, as características são organizadas hierarquicamente em forma de árvore e são unidas por relacionamentos estruturais formando agrupamentos. Cada característica possui um especificador próprio que a define como obrigatória, opcional ou alternativa. Características obrigatórias constituem a essência para a definição de um modelo funcional abstrato do domínio em análise e estão presentes em todas as aplicações derivadas desse modelo. As características opcionais podem ou não estar presentes em uma aplicação, de acordo com os requisitos do projeto. Já as características alternativas constituem um grupo do qual apenas uma é selecionada adequadamente para fazer parte de uma aplicação.

A Figura 5 mostra um modelo de características exemplificando as definições propostas por Kang et al. (1990).

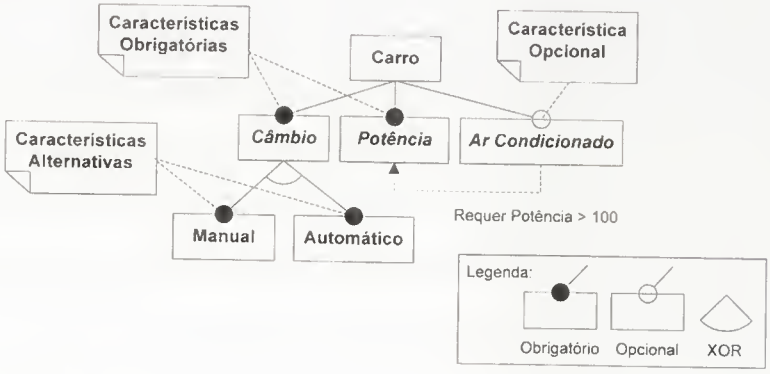


Figura 5. Modelo de Características [Adaptado de (KANG et al., 1990)]

Para ampliar a capacidade representativa do modelo de características proposto por (KANG et al., 1990), Riebisch et al. (2002) propõem o uso do conceito de multiplicidade presente na UML (BOOCH; RUMBAUGH; JACOBSON, 1999) que, aplicado aos relacionamentos entre características, permite representar suas possíveis combinações em um agrupamento. No exemplo da Figura 6 um leitor deve fornecer seus dados pessoais, que contêm obrigatoriamente seu nome e duas ou mais informações adicionais entre Endereço, Nascimento, RG e Nome da mãe.

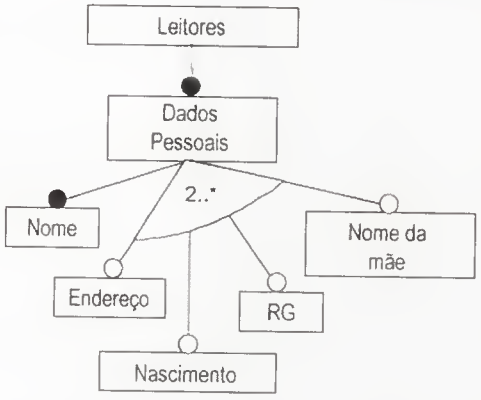


Figura 6. Modelo de características com uso do conceito de multiplicidade [Adaptado de (RIEBISCH et al., 2002)]

FORM (*Feature-Oriented Reuse Method*) (KANG et al., 1998) estende o método FODA (KANG et al., 1990) para a realização de Engenharia de Domínio, Figura 1. Esse método usa modelos de características, criados na fase de análise de domínio, para realizar as fases de projeto e implementação de domínio, produzindo uma arquitetura de domínio e um conjunto de componentes reutilizáveis, que são utilizados pela Engenharia de Aplicação para a implementação de uma família de produtos. Por meio de análise das características do domínio são feitas tentativas de mapeamento dessas para componentes arquiteturais. Cada característica é projetada como um componente distinto. Se houver dificuldade em estabelecer essa relação, uma característica pode ser redefinida em características mais específicas, que podem ser mais facilmente mapeadas em componentes arquiteturais. Para Sochos, Philippow e Riebisch (2004) esse método é vago em relação ao processo de mapeamento característica-arquitetura, ocasionando freqüentes divisões e junções de características, como ilustrado na Figura 7.

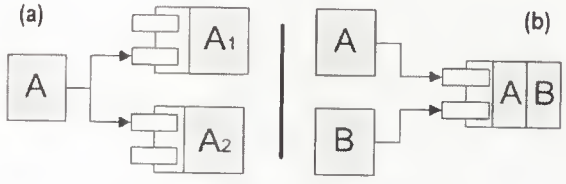


Figura 7. Divisão da característica A em dois componentes A1 e A2 (a); Junção das características A e B em um componente AB (b) [Adaptado de (SOCHOS; RIEBISCH; PHILIPPOW, 2004)]

Após analisarem as deficiências observadas, Sochos, Riebisch e Philippow (2006) apresentam o método FArM (*Feature-Architecture Mapping*). Como mostra a Figura 8,

FArM deve ser aplicado após a fase de Análise de Domínio para o desenvolvimento do núcleo de ativos para LPS.

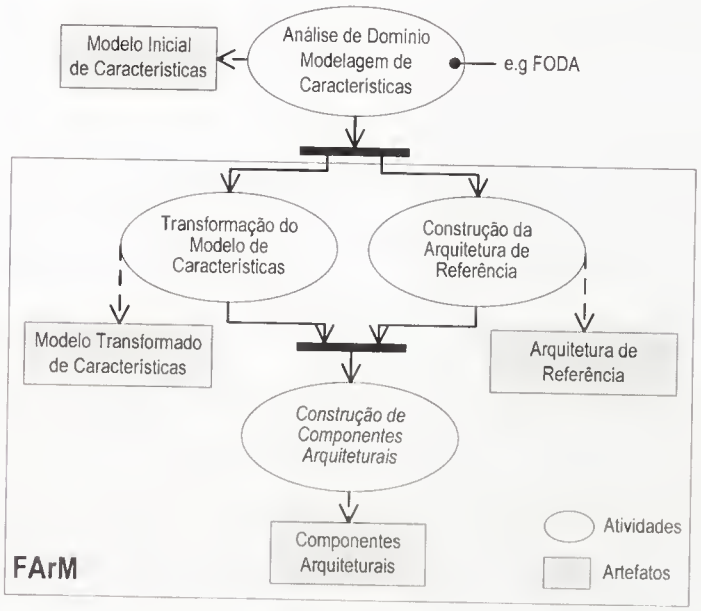


Figura 8. Método FArM [Adaptado de (SOCHOS; PHILIPPOW; RIEBISCH, 2006)]

Após a elaboração de um modelo inicial de características ocorrem duas atividades: Transformação do Modelo de Características e Construção da Arquitetura de Referência. Um Modelo Transformado de Características, compatível com uma recém criada Arquitetura de Referência do tipo *plug-in*, são os ativos resultantes. Cada característica identificada é implementada como um componente arquitetural único (componente-característica) pela atividade de Construção de Componentes Arquiteturais. Posteriormente, na Engenharia de Aplicação, se os requisitos do cliente podem ser satisfeitos pelo conjunto de características implementadas, a geração do produto é feita por adições de componentes-características à plataforma *plug-in*, cujas ações são estabelecidas pelo conjunto de parâmetros fornecidos.

2.8 Manutenção e Reengenharia de Software

Manutenção de software é o conjunto de atividades necessário para prover suporte a um sistema ao longo do seu ciclo de vida (PIGOSKI, 1997), *i.e.*, corrigir erros e falhas de requisitos e de projeto, implementar melhorias, adaptar códigos para diferentes tipos de hardware etc. Manutenções podem ser classificadas como: a) Corretiva: realizada para corrigir falhas encontradas na aplicação; b) Adaptativa: realizada para manter a aplicação

utilizável mediante modificações do ambiente; c) Perfectiva: realizada para melhorar o desempenho, inserindo novas funções, e a manutenibilidade da aplicação; d) Preventiva: realizada para detectar e corrigir falhas latentes, facilitando manutenções futuras e e) Emergencial: Manutenção Corretiva não programada (IEEE STD, 1998).

Em condições ideais, manutenções não deveriam degradar a confiabilidade e a estrutura de sistemas, nem diminuir seu grau de manutenibilidade. De outra forma estariam onerando e dificultando progressivamente a realização de futuras alterações. Na prática, porém, o que se observa é o fenômeno de envelhecimento dos sistemas após sucessivas manutenções (PARNAS, 1994).

Um dos modelos usados para a realização de manutenções é denominado *quick-fix*, Figura 9, no qual o código é alterado prioritariamente, sendo em seguida atualizada toda a documentação associada, requisitos, análise, projeto e testes. Na prática, muitas vezes por escassez de tempo e recursos financeiros, modificações ocorrem sem planejamento, projeto, análise de impacto ou testes de regressão. Nesses casos, a documentação, quando existente, pode não ser atualizada apropriadamente, refletindo a modificação feita, ocasionando sua degradação. Modelos evolutivos, Figura 10, constituem uma alternativa para a manutenção de software e propõem que sistemas sejam construídos progressivamente por meio de implementações que completam, corrigem e refinam os requisitos das anteriores (CANFORA; CIMITILE, 2000). Basili (1990) apresenta um modelo de processo no qual a manutenção é vista como um caso particular de desenvolvimento de software orientado para o reúso, Figura 11, cuja idéia central é a criação de um repositório comum de artefatos para sistemas de um mesmo domínio. Assim, a implementação de novos sistemas pode ser realizada com menor custo e esforço, utilizando-se desse conjunto de artefatos, em diferentes níveis de abstração.

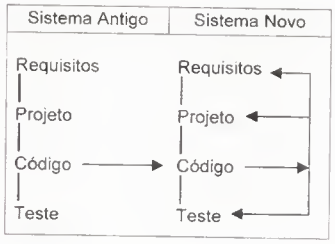


Figura 9.
Modelo Quick-Fix

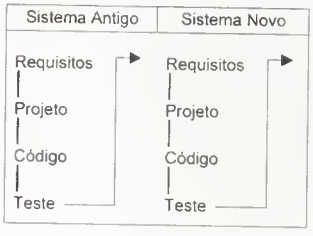


Figura 10.
Modelo Evolutivo

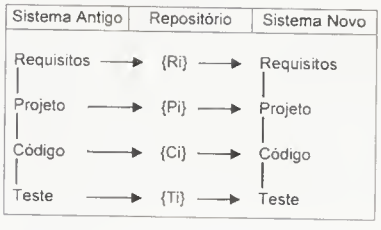


Figura 11.
Modelo com Reúso

[Adaptados de (CANFORA; CIMITILE, 2000)]

Diferentes modelos de processos têm sido propostos para a realização de manutenção de software, cada qual com um conjunto particular de fases ou tarefas seqüenciais. O modelo proposto pelo IEEE (IEEE STD, 1998), por exemplo, define as seguintes fases: a) Identificação, classificação e priorização do problema ou alteração: a solicitação de alteração é recebida e analisada em relação à possibilidade de ser realizada, podendo ser aceita ou rejeitada. Se aceita, recebe uma identificação e uma prioridade; b) Análise: são criados planos preliminares para as fases de projeto, implementação, testes e liberação c) Projeto: a documentação de projeto é criada juntamente com casos de teste; d) Implementação: a modificação é implementada e o código integrado à aplicação; e) Testes de Regressão: todo o sistema é testado para assegurar a conformidade da implementação com os requisitos originais; f) Testes de Aceitação: um teste mais amplo é realizado para validação do sistema e g) Liberação: o sistema testado é liberado para instalação e uso.

Apesar das diferentes propostas existentes, há um consenso em relação às atividades consideradas indispensáveis para o sucesso do processo de manutenção. São elas: compreensão do código existente, análise de impacto das mudanças propostas e testes de regressão (CANFORA; CIMITILE, 2000).

A reengenharia incremental constitui uma técnica alternativa para a manutenção evolutiva de software por meio de iterações, que pode ser desenvolvida por meio de dois métodos: 1) Com reintegração: módulos já tratados são reintegrados ao sistema legado sem alteração de linguagem ou paradigma de programação; 2) Sem reintegração: módulos já tratados não são reintegrados ao sistema legado, mas permanecem conectados a ele por meio de *gateways*[6] (O'BRIEN, 2005). Uma de suas vantagens é a atualização gradativa da documentação em concordância com as modificações feitas no código. Outras vantagens dessa técnica são: resultados concretos e imediatos, menor risco, melhor recuperação de erros e aumento de flexibilidade. Um ponto chave para a realização da reengenharia incremental é definir a forma de decompor o sistema legado em seus elementos constituintes. Existem cinco métodos para fazê-lo: Estrutural (Componentes do Hardware), Funcional (Funções Estratégicas), por Grau de Interconectividade (Sistemas não Críticos), Dados e por Combinação desses (OLSEM, 1998).

[6] Código que conecta dois módulos de software em diferentes estágios evolutivos.

Reengenharia de Software foi definida por Chikofsky e Cross (1990) como um processo constituído de Engenharia Reversa, na qual um projeto de software é recuperado a partir de seus artefatos legados, seguida de Engenharia Avante, na qual o software é reconstruído com tecnologias mais modernas, porém, preservando sua funcionalidade original.

Para Pressman (2002), engenharia reversa é o processo de análise de um programa, em um esforço para representá-lo em um nível de abstração mais alto do que o código-fonte, Figura 12. Ferramentas de engenharia reversa podem ser usadas para extrair informações do projeto de dados, arquitetural e procedimental de um programa existente. A engenharia avante não apenas recupera as informações de projeto de um software existente, mas também as utiliza para alterar ou reconstruir o sistema existente, em um esforço para aperfeiçoar sua qualidade global. Na maioria dos casos, o software tratado pela reengenharia reimplementa as funções originais do sistema existente, adiciona a ele novas funções e melhora seu desempenho de forma geral.

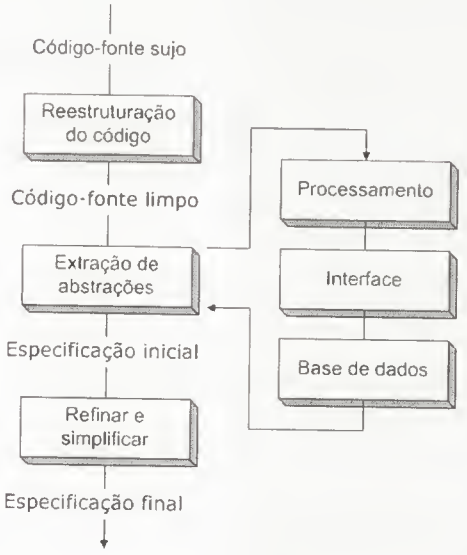


Figura 12. Processo de engenharia reversa [Adaptado de (PRESSMAN, 2002)]

Tilley e Smith (1996) mencionam que em qualquer reengenharia de sistemas legados, é importante considerar características cognitivas ligadas ao entendimento de seu código, que abrangem uma combinação de estratégias, classificadas da seguinte forma:

- Bottom-Up: Reconstrói o projeto de alto nível de um sistema a partir de seu código fonte, por meio de uma série de fragmentos e passos de atribuição de conceitos.

- Top-Down: Começa com prévia noção da funcionalidade do sistema e assinala componentes individuais deste, responsáveis por tarefas específicas.
- Refinamento iterativo: Cria, verifica e modifica hipóteses até que o sistema todo esteja retratado por um conjunto consistente de hipóteses.
- Combinação: explora oportunamente as estratégias Top-Down e Bottom-Up.

O'Brien (2005) menciona cinco estratégias para a reengenharia de sistemas: descartá-los, reescrevê-los do zero, integrá-los nas aplicações atuais e futuras (com *wrappers* e *gateways*), *data warehouse* e migração gradativa. Afirma, ainda, que em uma migração gradativa, sistemas legados e sistemas que já passaram por reengenharia podem coexistir durante o estágio de migração. O desafio está no projeto de *gateways* para isolar os passos da migração, de modo a tornar imperceptível aos usuários do sistema o fato do código e dos dados que se têm acesso residirem no sistema antigo ou no novo.

A migração gradativa é explorada por Bianchi et al. (2003) que apresentam uma abordagem de reengenharia iterativa baseada em diferentes repositórios para artefatos legados, transientes e definitivos. A movimentação dos artefatos de um repositório para outro é feita gradativamente e iterativamente, *i.e.*, em intervalos de tempo pequenos e freqüentes, até que todos tenham sido movidos para o repositório de artefatos definitivos. Os autores afirmam que esse processo permite a reengenharia de sistemas sem congelamentos ou duplicações de artefatos.

Kang et al. (2005) propõem um processo de reengenharia orientado a características para criação de LPS a partir de sistemas legados, cuja visão geral é mostrada na Figura 13. Primeiramente são extraídos do código legado componentes e informações da arquitetura. As características das aplicações são posteriormente modeladas com base nas informações recuperadas e no conhecimento do domínio. Antecipando evoluções futuras dessas aplicações, considerando oportunidades potenciais de negócio, mudanças de tecnologia etc., são feitos refinamentos no modelo de características obtido, no qual são inseridas características adicionais e informações de suas variabilidades. Finalmente, baseado nesse modelo refinado são produzidos a nova arquitetura e os componentes da LPS.

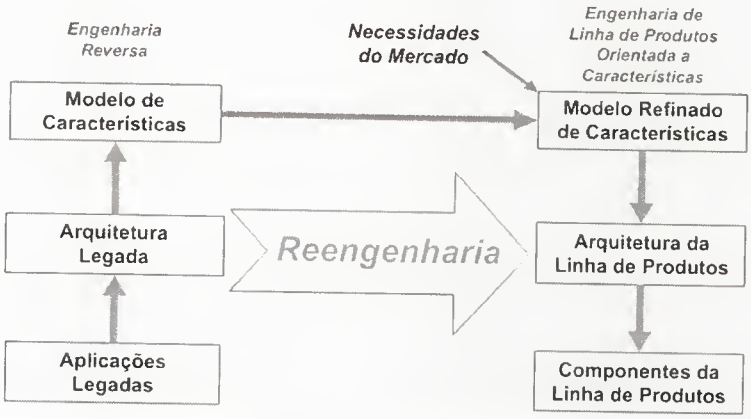


Figura 13. Visão geral do processo de reengenharia orientado a características [Adaptado de (KANG et al., 2005)]

Componentes de LPS podem ser obtidos por técnicas de mineração, Seção 2.6, a partir das aplicações legadas. Bergey, O'Brien e Smith (2000) propõem a realização da mineração de ativos para LPS em quatro passos: 1) Reunir as informações preliminares, 2) Decidir pela mineração e definir sua estratégia, 3) Obter entendimento técnico detalhado dos ativos existentes e 4) Recuperar os ativos. Bosch e Ran (2000) consideram essencial a criação de um modelo de características (KANG et al., 1990) para apoiar a realização dessa tarefa.

2.9 Considerações Finais

Para Stojanovic, Dahanayake e Sol (2003) várias metodologias ágeis adotam o paradigma de desenvolvimento orientado a objetos sem usarem conceitos mais avançados de software, como, por exemplo, componentes (HEINEMAN; COUNCILL, 2001). Em sua opinião, componentes como provedores de serviços, que elevam o nível de abstração das tradicionais classes/objetos, podem apoiar princípios e práticas do desenvolvimento ágil como: simplicidade, flexibilidade para mudanças, boa comunicação entre membros de uma equipe e rápido *feedback*. O uso de componentes permite ainda que um problema seja dividido em partes menores, que contenham um conjunto coeso de regras de negócios, provendo uma visão mais abstrata do sistema, facilitando a definição de prioridades e a formação de equipes especializadas. Radinger e Goeschka (2003) descrevem uma forma de integrar metodologias ágeis em processos de Engenharia de Software Baseada em Componentes (ESBC), propondo o uso dessas para o desenvolvimento de componentes, cujos requisitos advêm de artefatos criados em fases do processo anteriores à de implementação.

A mineração de componentes a partir de códigos legados e a sua posterior reintegração ao sistema original é uma técnica que pode não só modernizar o código existente, mas também estender suas funções, garantindo-lhe uma sobrevida (MEHTA; HEINEMAN, 2002). Essa possibilidade é particularmente importante para o domínio de sistemas embutidos, no qual o núcleo de uma aplicação legada geralmente passa por várias manutenções ao longo do seu ciclo de vida, para acompanhar as evoluções do hardware.

A busca de empresas por técnicas de desenvolvimento de software, que otimizem processos de desenvolvimento de aplicações em seus respectivos domínios, tem favorecido a proliferação de LPS (MYLLYMÄKI; KOSKIMIES; MIKKONEN, 2002). Para eles, embora o conceito de LPS tenha sido amplamente adotado, o seu projeto de arquitetura (BOSCH, 2000) é ainda uma difícil tarefa. Tipicamente, projetos de LPS usam técnicas variadas como: estilos arquiteturais (SHAW; GARLAN, 1996), componentes (HEINEMAN; COUNCILL, 2001), *frameworks* (FAYAD; SCHMIDT; JOHNSON, 1999), padrões de projeto (GAMMA et al., 1995) e programação generativa (CZARNECKI; EISENECKER, 2000). Assim, não fica claramente definido como uma plataforma de desenvolvimento de LPS deve ser estruturada, que tipo de variabilidade é suportado pelas diferentes partes da plataforma, que tipo de especialização ou interface de uso elas oferecem, e como são feitas atribuições de versões para essas partes. Um estudo comparativo entre diferentes abordagens de LPS concluiu que elas parecem não competir entre si, pois apresentam características particulares distintas (MATINLASSI, 2004), evidenciando a falta de entendimento em relação aos elementos constituintes de LPS (MYLLYMÄKI; KOSKIMIES; MIKKONEN, 2002). Estudos como esse podem facilitar processos de escolha entre diferentes abordagens.

Desenvolvimento ágil de aplicações e LPS são técnicas que, embora tenham o objetivo comum de melhorar a produtividade de software, possuem requisitos aparentemente contraditórios, que dificultam seu uso conjunto. Princípios ágeis enfatizam simplicidade e velocidade, com implementação de versões incrementais de requisitos atuais. Já LPS prevê um planejamento inicial de todos os produtos que podem ser derivados dela, que serão desenvolvidos com base em uma arquitetura comum e um núcleo de ativos reutilizáveis previamente projetados (PAIGE et al., 2006).

É desejável que práticas de reengenharia tratem não só da modernização de sistemas legados, mas também de sua evolução para um núcleo comum de artefatos reutilizáveis, que

possa alicerçar o desenvolvimento futuro de software de alta qualidade, mais rápido, com maior produtividade e eficiência (WEIDERMAN et al., 1997). Alguns sistemas legados podem não permitir o congelamento total ou parcial de suas tarefas durante um processo de reengenharia. Apesar de Bianchi et al. (2003) enfatizarem que seu processo permite a reengenharia iterativa de sistemas legados sem congelamentos ou duplicações de artefatos, o modelo proposto por eles contraria a idéia de Weiderman et al. (1997) e promove alterações de software no nível de instruções em uma linguagem de programação de alto nível, e não no nível arquitetural. No entanto, sua proposta de uso de diferentes repositórios, para armazenar separadamente artefatos em estágios diferentes de migração, pode ser explorada para uma possível integração com LPS, permitindo a coexistência de código legado e de ativos da LPS durante o processo de reengenharia.

Capítulo 3

Reengenharia Iterativa Orientada a Características

3.1 Considerações Iniciais

Modernas técnicas e tecnologias de engenharia de software para o desenvolvimento de novos produtos não são atraentes para a maioria das empresas devido à complexidade em utilizá-las, ao baixo grau de compatibilidade com a base legada e a pouca maturidade de suas ferramentas (GRAAF; LORMANS; TOETENEL, 2003).

A maioria dos projetos reais não começa do zero. Empresas têm sempre que lidar com código legado, o que significa que novas tecnologias devem ter um grau mínimo de compatibilidade com aquelas anteriormente utilizadas. Adicionalmente, empresas desejam utilizar partes de seus produtos bem sucedidos para compor novos produtos com pouca ou nenhuma adaptação. Essa forma de desenvolvimento sugere que abordagens *botton-up* ou combinada sejam utilizadas. Porém, várias das recentes técnicas de engenharia de software adotam abordagens *top-down* (GRAHAM, 2000).

Novas tecnologias de desenvolvimento são geralmente concebidas de maneira conceitual, não sendo fornecidos, no início, muitos detalhes sobre as possíveis formas de implementá-las. Quando passam a ser apoiadas por ferramentas, essas não correspondem às expectativas das empresas na condução de soluções particulares ou complexas, devido à sua pouca maturidade.

Técnicas complexas de desenvolvimento requerem o envolvimento de engenheiros de software experientes para colocá-las em prática. Além disso, muitos de seus elementos técnicos podem não ser compreendidos por pessoas de outras áreas, que habitualmente integram ambientes reais de desenvolvimento. Para serem aceitas pelas empresas, novas tecnologias devem ser facilmente aplicáveis ou disporem de meios para ocultar as suas reais complexidades.

Esses princípios de aceitabilidade são considerados pela abordagem de reengenharia iterativa orientada a características, proposta neste trabalho, que une princípios ágeis a

técnicas modernas de desenvolvimento e de manutenção de software voltadas para o reúso. Assim, é possível realizar a reengenharia de sistemas embutidos legados, obtendo a sua revitalização, concomitantemente com a criação de um núcleo de artefatos reutilizáveis, para apoio ao desenvolvimento ágil de produtos similares, membros de uma família de produtos de software (CLEMENTS; NORTHROP, 2001).

A Seção 3.2 contextualiza as diretrizes adotadas para a criação da abordagem de reengenharia iterativa orientada a características; a Seção 3.3 detalha a sua modelagem, descrevendo as atividades e os artefatos utilizados e produzidos e a Seção 3.4 apresenta as considerações finais.

3.2 Contextualização das Diretrizes Adotadas para a Criação da Abordagem de Reengenharia Iterativa Orientada a Características

A evolução do hardware de sistemas embutidos para produtos de consumo ocorre de maneira dinâmica a fim de atender necessidades imediatas do mercado. Nesse cenário é muito difícil antecipar as possíveis variações de um produto ao longo do seu ciclo de vida.

Empresas que desenvolvem software embutido geralmente não o distribui como produto, mas como parte integrante de seus equipamentos manufaturados, por exemplo, celulares e *palmtops*. Em muitas dessas empresas a engenharia de produtos ainda é fundamentalmente guiada pelo hardware, sob o ponto de vista mecânico e eletrônico. Nesses casos, engenheiros de software freqüentemente não participam de decisões de projeto na fase de concepção de novos produtos, ou de variações dos já existentes, e têm suas tarefas dificultadas, quando solicitados a realizar suas implementações ou manutenções (GRAAF; LORMANS; TOETENEL, 2003). Sob o ponto de vista desses profissionais, mudanças de requisitos devido a evoluções do hardware ocorrem de maneira não planejada. Esse fato dificulta o desenvolvimento de software embutido para a criação de produtos similares por meio de técnicas apoiadas por abordagens *top-down* e reúso planejado. Por outro lado, ele favorece o uso de métodos adaptados a mudanças e ágeis o suficiente para implementá-las de forma rápida, possivelmente por meio do reúso total ou parcial do código existente. Essa forma de reúso não se enquadra inteiramente em nenhuma das categorias definidas por Bosch

(2000), descritas na Seção 2.5, provavelmente por se tratar de uma necessidade de evolução particular de sistemas embutidos, que envolve variações do hardware.

A necessidade de adicionar rapidamente variabilidades não planejadas em uma aplicação pode ocasionar a degradação de seu código, aumentando a dificuldade de manutenção. Se o código for duplicado, total ou parcialmente, para a criação de produtos similares, essa dificuldade pode ser ainda maior, passando a exigir alterações sincronizadas que podem não ser acompanhadas de uma documentação adequada. Um processo de reengenharia para modernizar o código e melhorar a maneira de apoiar as variabilidades do domínio deve envolver não apenas a parte técnica, mas também a humana, pois uma empresa pode rejeitar o uso de novos recursos, com os quais seus profissionais não estão familiarizados. Detalhes dessa natureza, se não considerados no planejamento da reengenharia podem inviabilizar a sua implantação na prática.

Maximizar o reúso e apoiar variabilidades do domínio para a criação de produtos similares são requisitos que podem ser atendidos com o apoio de técnicas de LPS. Para algumas delas, porém, a ocorrência de variabilidades não planejadas pode ser um problema. Por exemplo, no método FAST (WEISS; LAI, 1999), Figura 2, variabilidades não apoiadas pelo Ambiente de Engenharia de Aplicação precisam ser desenvolvidas em uma nova iteração da Engenharia de Domínio. Se isso ocorrer de forma constante, a duração da fase de investimento pode ser prolongada indefinidamente, inviabilizando economicamente a manutenção da LPS. Outro fato a ser considerado é que o reúso citado pelos métodos de LPS anteriormente mencionados se apóia nas categorias planejado e *top-down*, definidos por Bosch (2000), considerando a base legada apenas como uma possível fonte para entendimento do domínio e mineração de artefatos reutilizáveis, contrariando os princípios de aceitabilidade propostos por Graaf, Lormans e Toetenel (2003). A criação de aplicações no método KobrA (ATKINSON et al., 2001), Figura 3, exemplifica essas categorias de reúso ao evoluir um sistema de uma caixa preta genérica e abstrata para uma árvore de componentes detalhados. Adicionalmente, é importante observar que nenhum dos dois métodos de LPS mencionados estabelece uma forma clara para criar uma LPS a partir dos artefatos legados de um domínio. Esse fato pode ser um dos motivos para a falta de processos de reengenharia que utilizam esses métodos. Por outro lado, modelos de características (KANG et al., 1990) têm apoiado a migração de aplicações legadas para LPS por meio de processos de reengenharia, como o de

Kang et al. (2005), Figura 13. Esses autores, além de considerarem a existência de aplicações legadas, também apóiam a proposta de evoluir o código dessas aplicações para um núcleo comum de artefatos reutilizáveis (WEIDERMAN et al., 1997). Esses processos, porém, propõem a realização integral de tarefas potencialmente complexas, como a recuperação da arquitetura legada, antes que uma única variabilidade, mesmo que simples, seja disponibilizada na forma de uma aplicação executável, o que contraria os princípios ágeis de desenvolvimento. A realização parcial dessas tarefas em curtas e freqüentes iterações, voltadas para atender os requisitos imediatos de evolução das aplicações legadas, pode ser uma das soluções para tornar ágeis os processos de reengenharia orientados a características voltados para LPS. Ressalta-se que o resultado final desse processo pode não ser uma LPS tradicional completa, *i.e.*, um modelo do domínio, uma arquitetura de referência e um núcleo de componentes arquiteturais genéricos, porém, uma forma mais simples, que atenda eficientemente às necessidades do domínio de sistemas embutidos, descritas anteriormente.

A entrega de versões parciais executáveis é um dos requisitos contraditórios entre princípios ágeis e LPS (PAIGE et al., 2006). Uma aplicação executável em uma Linha de Produtos de Software é o produto resultante da Engenharia de Aplicação e sua implementação só é possível ao final da Engenharia de Domínio, quando todo o ambiente de desenvolvimento, *i.e.*, ferramentas e artefatos, está disponível. Adicionalmente, o processo genérico de engenharia para LPS (ZIADI; JÉZÉQUEL; FONDEMENT, 2003), Figura 1, não evidencia como os componentes produzidos pela Engenharia de Domínio podem ser testados antes de integrarem uma aplicação real. A realização tardia de testes funcionais desses componentes pode aumentar a complexidade dos testes de aceitação da aplicação e também ocasionar recursões à Engenharia de Domínio, onerando o processo de desenvolvimento. Uma possível solução para esses problemas é observá-los sob o ponto de vista de reengenharia e não de um processo de engenharia avante.

O reúso integral do código fonte de uma aplicação agiliza a criação de versões similares executáveis, que apóiam variabilidades do domínio, e permite também validar, em ambiente real, os novos componentes desenvolvidos. A aplicação existente pode ser vista como sendo a junção de todas as características conhecidas do domínio em único componente auto-suficiente. Assim, é possível planejar a decomposição incremental desse componente (OLSEM, 1998) para criar componentes-característica (SOCHOS; RIEBISCH; PHILIPPOW,

2006) menores, mais especializados e que apóiam variabilidades do domínio por meio de parametrizações (KRUEGER, 2002). Essa tarefa pode ser apoiada por modelos de características, propostos por Kang et al. (1990). Após serem criados, esses componentes voltam a se conectar ao componente principal, porém, por meio de *gateways* (O'BRIEN, 2005), que promovem as adaptações de interface necessárias.

Os prováveis benefícios do processo mencionado são:

- Modernização gradativa do código da aplicação com o uso de componentes;
- Apoio rápido a novas variabilidades do domínio por meio de componentes alternativos ou parametrizáveis;
- Teste do componente recém-criado em ambiente real, no domínio da aplicação existente.

A modelagem da abordagem de reengenharia iterativa proposta neste trabalho é apresentada a seguir e baseia-se na contextualização e nas proposições fornecidas nesta seção.

3.3 Modelagem do Processo de Reengenharia Iterativa Orientada a Características

O Processo de Reengenharia Iterativa aplica-se a sistemas embutidos legados que, devido à sua importância estratégica e obsolescência tecnológica ou funcional, precisam ser rapidamente revitalizados. Durante a sua realização, pode-se recorrer freqüentemente à documentação existente e ao código legado para a identificação e a compreensão de estruturas, funções e comportamentos dos sistemas. Os desenvolvedores de sistemas embutidos, aqui chamados de especialistas do domínio, têm a responsabilidade de divulgar os conhecimentos específicos do domínio, centralizar a origem das informações, auxiliar na compreensão do código legado, relatar as deficiências atuais dos sistemas e predizer, quando possível, suas necessidades futuras. A Figura 14 apresenta uma visão geral do Processo de Reengenharia Iterativa proposto.

Para a modernização incremental de sistemas legados, as fases de Engenharia Reversa e Avante são realizadas aos pares, iterativamente. O processo pode ser cancelado ou interrompido temporariamente durante a sua realização, produzindo artefatos parcialmente modernizados, resultantes de uma Reengenharia Parcial. Essa, por sua vez, pode ser antecipadamente planejada para, por exemplo, implementar apenas um conjunto reduzido de variabilidades pré-determinadas pelo cliente, oferecendo-lhe rapidamente uma ou mais

variações de um produto, para que possa atender a potenciais oportunidades de negócios. Nesse caso, a Reengenharia Parcial pode ser considerada completa, no sentido de ter alcançado o objetivo do projeto, mesmo que tenha ocorrido somente a modernização parcial dos artefatos legados. O processo pode ser retomado, posteriormente, para ampliar a gama de produtos existentes. Se houver mais de uma aplicação legada similar no domínio e o objetivo da reengenharia for, prioritariamente, a criação de um conjunto de artefatos reutilizáveis, essas podem ser tratadas concomitantemente. Dessa forma, é possível melhorar o grau de generalidade dos artefatos produzidos para o apoio às suas similaridades.

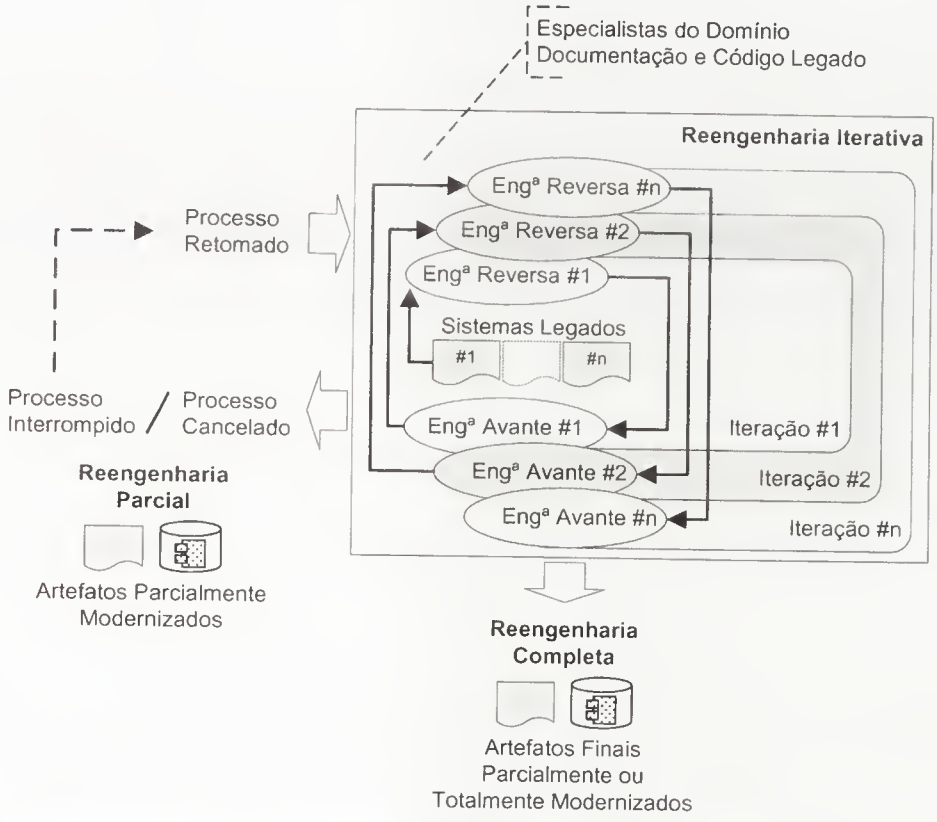


Figura 14. Visão geral do Processo de Reengenharia Iterativa

As próximas subseções detalham as fases de Engenharia Reversa e Avante do Processo de Reengenharia Iterativa proposto. Modelos de atividades adaptados da UML (BOOCH; RUMBAUGH; JACOBSON, 1999) são usados para ilustrar as atividades realizadas e os artefatos utilizados e produzidos em cada uma delas.

3.3.1 Engenharia Reversa

A fase de Engenharia Reversa é responsável pela criação e refinamento de abstrações dos sistemas. Na sua primeira iteração são criadas as primeiras abstrações do sistema legado a partir dos artefatos disponíveis, com o apoio de Especialistas do Domínio (ED).

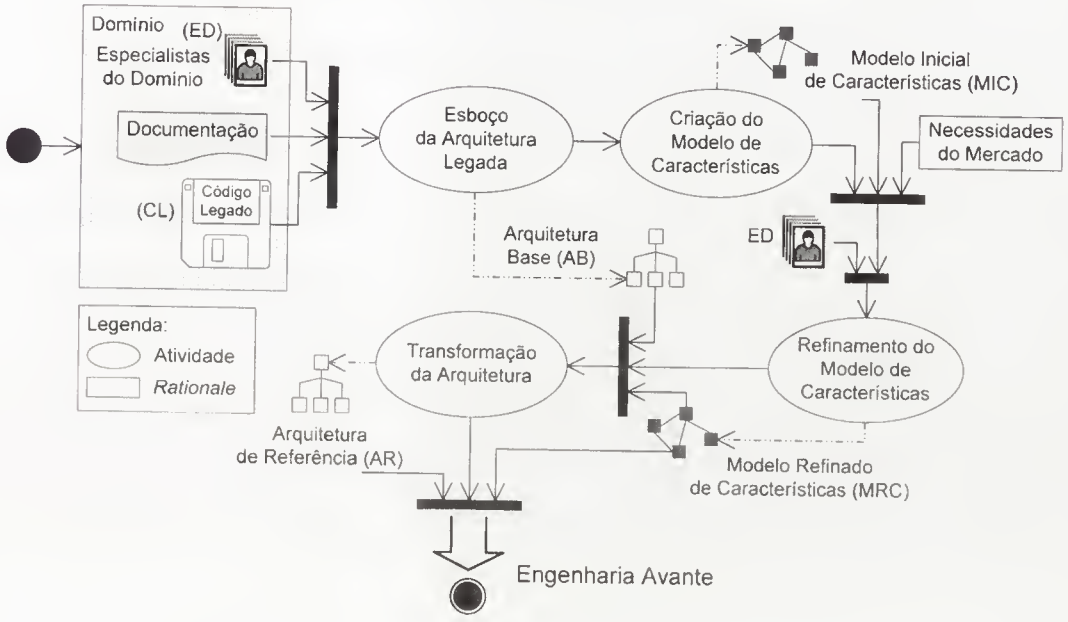


Figura 15. Primeira iteração da Engenharia Reversa

O Esboço da Arquitetura Legada e a Criação do Modelo de Características são as primeiras atividades realizadas e apóiam-se na proposta de engenharia reversa de Kang et al. (2005), Figura 13. Seus resultados são respectivamente uma Arquitetura Base (AB) e um Modelo Inicial de Características (MIC) do domínio. Porém, sem uma definição inicial das partes do sistema que serão tratadas pela reengenharia e suas prioridades, a Arquitetura Base deve conter somente um esboço e não uma modelagem detalhada da arquitetura legada, fornecendo apenas uma macrovisão do sistema, por meio de blocos estruturais e seus inter-relacionamentos. Da mesma forma, o Modelo Inicial de Características não deve conter características detalhadas do domínio, mas sim um conjunto de macrocaracterísticas relacionadas, que o represente de forma completa em um alto nível de abstração. Essa etapa requer a participação de Especialistas do Domínio na criação e validação dos artefatos produzidos. Essa agilidade inicial permite o rápido envolvimento do cliente no processo, antes que artefatos mais detalhados, e possivelmente desnecessários, comecem a ser produzidos. Sucessivas iterações permitem a criação e o refinamento incremental dos mesmos em

momentos mais adequados, apoiados por um conhecimento mais sólido do sistema e do domínio, adquirido ao longo do processo.

A partir da obtenção da **A**rquitetura **B**ase e do **M**odelo **I**nicial de **C**aracterísticas é realizado um conjunto de atividades iterativas de manutenção, que seguem a proposta do IEEE Std (1998). São elas: a) Identificação, classificação e priorização do problema ou alteração: Necessidades do Mercado são captadas dinamicamente por pessoas envolvidas direta ou indiretamente com o sistema, facilitando a identificação de características estratégicas do domínio e a definição de suas prioridades e b) Análise: as características identificadas são adicionadas a uma representação do domínio e recebem um plano preliminar para seu projeto, implementação, teste e liberação.

A atividade de Refinamento do Modelo de Características provê, a cada iteração e com o auxílio de **E**specialistas do **D**omínio, a adição de novas características ao modelo atual, o detalhamento de características abstratas existentes nesse modelo e o refinamento de características identificadas como sendo a junção de outras mais específicas (SOCHOS; PHILIPPOW; RIEBISCH, 2004). Seu resultado é um **M**odelo **R**efinado de **C**aracterísticas (MRC), que modela incrementalmente o domínio, priorizando a representação de características estratégicas, segundo as Necessidades do Mercado. Com base na **A**rquitetura **B**ase e no **M**odelo **R**efinado de **C**aracterísticas recém criado a realização da atividade de Transformação Arquitetural resulta em uma **A**rquitetura de **R**eferência (AR) (SOCHOS; PHILIPPOW; RIEBISCH, 2006) que deve retratar os novos elementos arquiteturais presentes na estratégia adotada para a reengenharia do sistema, como por exemplo, componentes (HEINEMAN; COUNCILL, 2001) e *gateways* (O'BRIEN, 2005). Se essa estratégia prevê o reúso total ou parcial do código legado, a **A**rquitetura de **R**eferência pode ser modelada como uma extensão mais detalhada da **A**rquitetura **B**ase.

Nas demais iterações da Engenharia Reversa, Figura 16, a atividade de Refinamento do Modelo de Características é realizada repetitivamente para retratar novas Necessidades do Mercado, melhorando a representatividade do **M**odelo **R**efinado de **C**aracterísticas em relação ao domínio. O resultado da atividade de Refinamento da Arquitetura após cada interação é uma **A**rquitetura de **R**eferência revisada para acomodar novos elementos arquiteturais, identificados ou criados, e para suprir eventuais deficiências observadas.

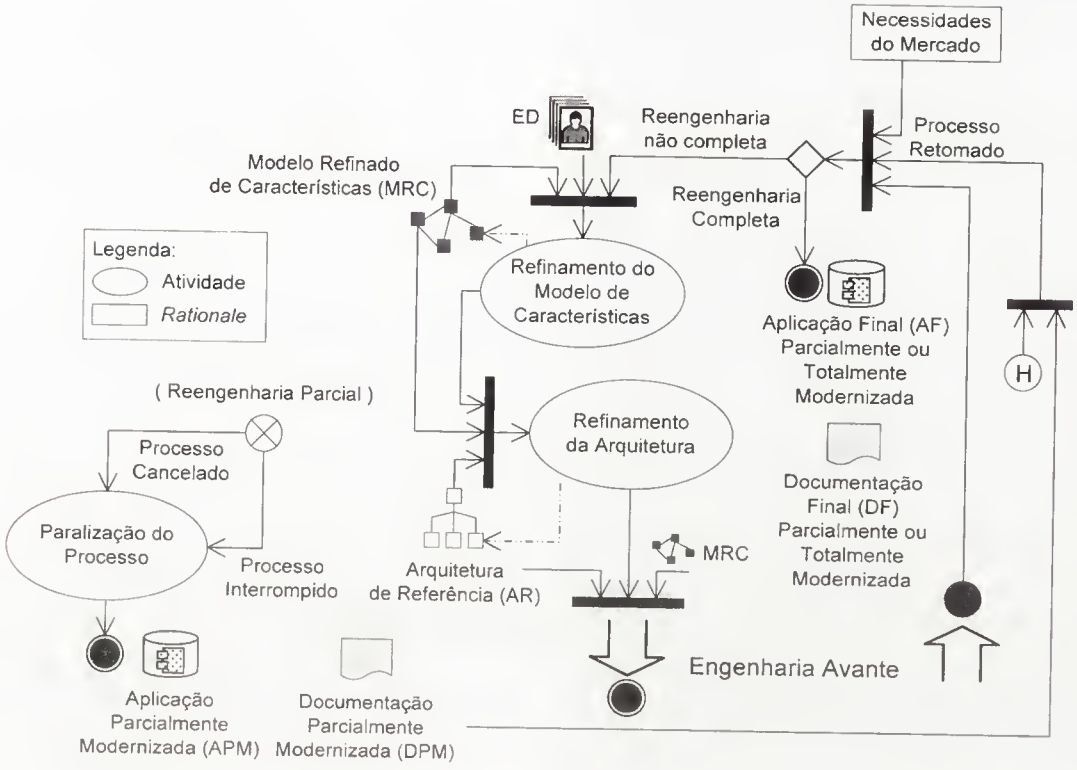


Figura 16. Demais iterações da Engenharia Reversa

A reengenharia é considerada completa quando todas as Necessidades do Mercado tiverem sido atendidas, o que não significa necessariamente, uma modernização completa do sistema legado, uma vez que a abordagem prevê a realização de reengenharia parcial planejada, como descrito anteriormente. Nesse caso, a atividade de Paralisação do Processo documenta os fatos associados à paralisação e prepara o ambiente e os artefatos atuais para uma posterior retomada do processo.

A Figura 17 apresenta a fase de Engenharia Reversa de forma completa, combinando as atividades e os artefatos comuns e particulares de cada iteração, e mostra como iteratividade e modelos de características são integrados no presente processo para a modelagem incremental do domínio e a modernização gradativa de seus artefatos.

A seção seguinte detalha as atividades da fase de Engenharia Avante que, como essa, também segue a proposta do IEEE Std (1998) para a realização de manutenções, com a realização das atividades de Projeto, Implementação, Testes e Liberação.

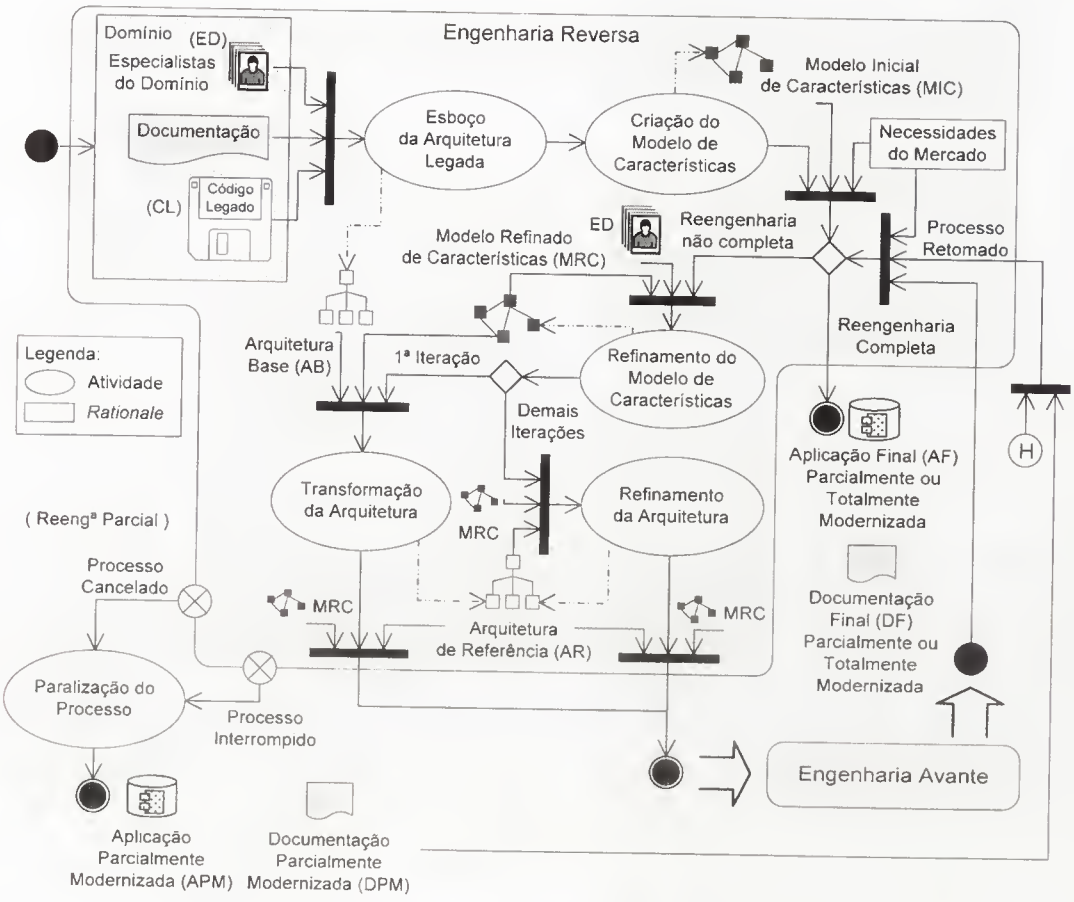


Figura 17. Atividades e artefatos da Engenharia Reversa

3.3.2 Engenharia Avante

Em concordância com os princípios ágeis de desenvolvimento, Seção 2.4, as atividades da fase de Engenharia Avante, Figura 18, são realizadas iterativamente e resultam em entregas parciais de versões incrementais modernizadas, executáveis e testadas do sistema ao cliente, na recuperação gradativa de sua documentação e na produção de um núcleo de artefatos reutilizáveis, para apoio ao desenvolvimento de produtos similares (WEIDERMAN et al., 1997).

A estratégia de modernização adota a proposta descrita ao final da Seção 3.2 e prevê o reúso integral do código legado (GRAAF; LORMANS; TOETENEL, 2003), do qual são extraídos componentes-características reutilizáveis (SOCHOS; RIEBISCH; PHILIPPOW, 2006), que, posteriormente, são re-conectados a ele por meio de *gateways* (O'BRIEN, 2005). Esses elementos são armazenados em repositórios distintos para permitir a realização dessas

atividades de maneira imperceptível para os atuais mantenedores do sistema (BIANCHI et al., 2003).

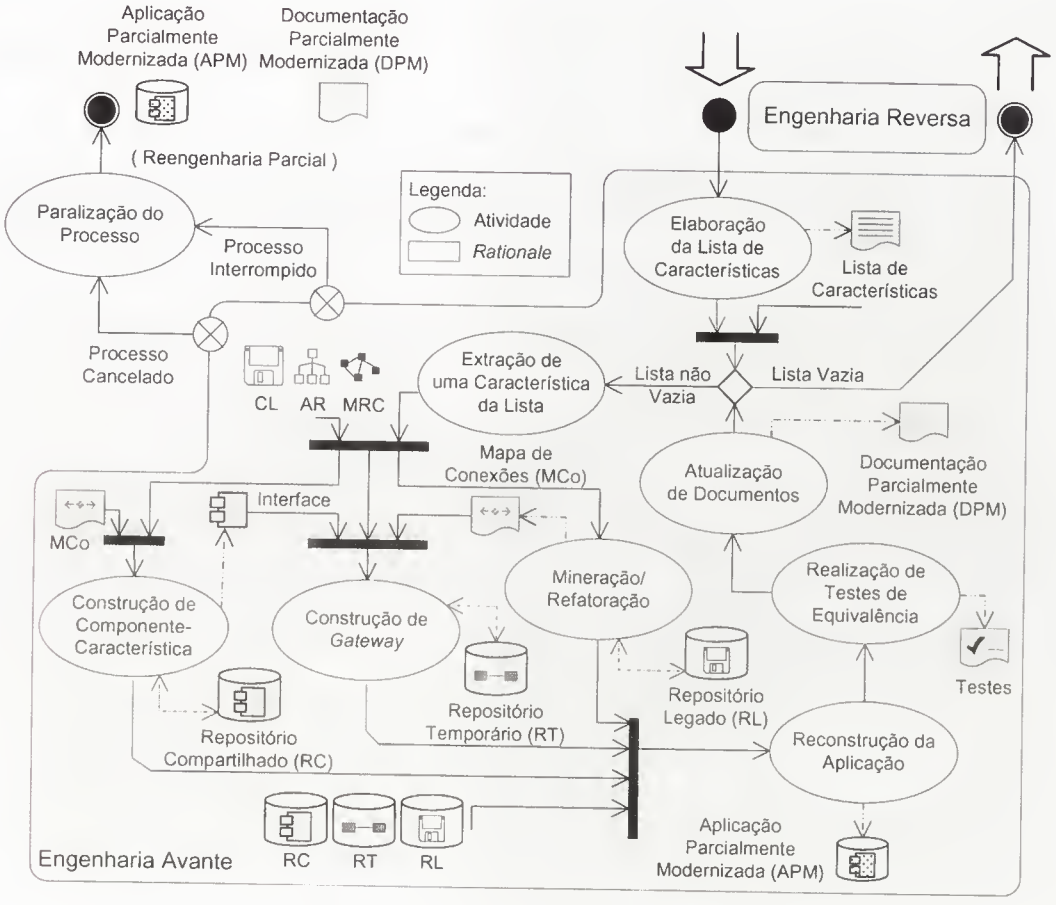


Figura 18. Atividades e artefatos da Engenharia Avante

A primeira atividade, Elaboração da Lista de Características, consiste em extrair uma **L**ista de **C**aracterísticas (LC), ainda não implementadas, a partir do **M**odelo **R**efinado de **C**aracterísticas atual, dando início a um processo iterativo de criação de componentes-características. A cada iteração é extraído um item dessa lista, *i.e.*, uma característica, com base nas prioridades do projeto, estabelecidas a partir das Necessidades do Mercado, dando início a três atividades: Mineração/Refatoração, Construção de Componente-Característica e Construção de *Gateway*.

Mineração/Refatoração é responsável por extrair do Código Legado (CL), armazenado no **R**epositório **L**egado (RL), a implementação da característica selecionada e por criar o seu **M**apa de **Co**nexões (MCo) com o código remanescente. Técnicas de Refatoração (FOWLER

et al., 2004) podem ser utilizadas para facilitar a extração de características a partir de códigos legados mal estruturados como, por exemplo, com forte acoplamento.

As colunas da Tabela 1 referem-se aos elementos que compõem o **M**apa de **Co**nexões e suas definições são dadas a seguir.

Tabela 1. Elementos do Mapa de Conexões de uma característica

Característica:		**<Identificação da característica >**			
Função	**Grupo**	**Parâmetros**	**Retorno**	**Comentários**	**Aplicações**
Nome da Função	Grupo da Função	Parâmetros da Função	Retorno da Função	Descrição da função e comentários relevantes	Aplicações que utilizam a Função
[Limitações] **[Novos Requisitos]** **[Observações]**					

- Identificação da característica: Nome da característica no **M**odelo **R**efinado de **C**aracterísticas. Para facilitar sua localização, uma identificação estendida pode ser opcionalmente fornecida, a qual deve incluir as características que precedem a atual no ramo de árvore em que essa se encontra. Por exemplo, uma característica B, filha de A, pode ser identificada por A□B;
- Nome da Função: Função do código legado que implementa integralmente ou parcialmente a característica selecionada;
- Grupo: Se ações similares são realizadas por funções distintas, presentes em diferentes aplicações, essas devem receber um mesmo identificador de grupo quando inseridas no **M**apa de **Co**nexões. A modernização do código deve prover uma única função para apoiar todas as variações do grupo, atendendo de modo unificado às necessidades de todas as aplicações associadas a ele;
- Parâmetros da Função: Lista de parâmetros aceitos pela função. O objetivo de cada parâmetro deve ser facilmente identificado a partir de sua descrição, que deve ser da forma <nome> | <tipo de dado> como na UML (BOOCH; RUMBAUGH; JACOBSON, 1999). Para funções sem parâmetros deve ser usado o identificador 'Nenhum';
- Retorno da Função: Segue a mesma regra de nomenclatura definida anteriormente para os Parâmetros da Função;
- Comentários: Contém uma descrição resumida da funcionalidade implementada e pode, adicionalmente, conter comentários relevantes para auxiliar sua extração do código legado;

- Aplicações: Cada aplicação legada similar do domínio recebe inicialmente um identificador único. Quando identificada uma função comum a duas ou mais dessas aplicações, seus identificadores devem ser listados nesse espaço da tabela. Isso ocorre com freqüência quando a técnica de reúso utilizada prevê o compartilhamento ou a duplicação de códigos. Essa lista indica se a extração da característica envolve múltiplas aplicações;
- Limitações: Durante a extração de uma característica, limitações de qualquer natureza, *e.g.*, técnica ou funcional, podem ser identificadas e devem ser documentadas nessa seção, para serem levadas ao conhecimento dos **E**specialistas do **D**omínio. Esses devem indicar a melhor solução para superá-las. Para isso, podem ser organizadas reuniões específicas para esse propósito a fim de que o problema seja analisado e solucionado em equipe;
- Novos Requisitos: Quando a solução para uma determinada limitação da característica selecionada deve residir no componente-característica, sua descrição é adicionada nessa seção, como requisito para a construção do componente.
- Observações: Área de texto livre para facilitar a comunicação entre membros de uma equipe que trabalham juntos na extração de uma característica. Pode registrar o status do andamento da tarefa, uma lista de problemas pendentes etc.

A criação do **M**apa de **Co**nexões propicia um maior entendimento do código legado por meio da realização de inspeções, que permitem avaliar o nível de acoplamento do mesmo com cada característica a ser extraída, facilitando a realização de análises de impacto. Uma forma alternativa para identificar as funções do código legado que implementam uma determinada característica é por meio de consultas à documentação gerada a partir dos testes de regressão realizados anteriormente (MEHTA; HEINEMAN, 2002). A presença de funções fracamente relacionadas no **M**apa de **Co**nexões pode indicar uma junção de características, o que requer uma revisão do **M**odelo **R**efinado de **C**aracterísticas.

A Construção de Componente-Característica usa a **A**rquitetura de **R**eferência, o **M**odelo **R**efinado de **C**aracterísticas e o **M**apa de **Co**nexões para implementar a característica selecionada na forma de um componente-característica reutilizável, armazenado no **R**epositório **C**ompartilhado (RC). Sua interface é genérica o suficiente para apoiar os requisitos dos diferentes sistemas similares existentes e as futuras variabilidades da característica implementada. Para isso, deve ser construído com o uso de técnicas que lhe

forneçam a flexibilidade necessária, como, por exemplo, a parametrização de variabilidades (KRUEGER, 2002) entre outras.

A **A**rquitetura de **R**eferência determina a maneira como cada componente deve ser reconectado ao código legado. O **M**odelo **R**efinado de **C**aracterísticas mostra o relacionamento da característica selecionada com as outras do domínio, que pode ser de existência, de dependência, de uso etc. Assim, a implementação de um componente-característica não deve ser iniciada sem que estejam implementadas as características que compõem seus pré-requisitos. Essa ordem é estabelecida com base no **M**odelo **R**efinado de **C**aracterísticas durante a Elaboração da Lista de Características. O **M**apa de **Co**nexões fornece os requisitos da interface do componente-característica, cujo projeto deve considerar não apenas a compatibilidade dessa com as funções do código legado, mas também as limitações e os novos requisitos identificados durante a realização da atividade de Mineração/Refatoração.

A atividade de Construção de *Gateway* usa o **M**apa de **Co**nexões da característica e procura compatibilizar as funções de sua tabela com os recursos oferecidos pela interface do componente-característica recém criado, resolvendo eventuais discrepâncias entre os mesmos. Uma vez criado o *gateway* de conexão entre os elementos do **R**epositório **C**ompartilhado e do **R**epositório **L**egado, esse é armazenado no **R**epositório **T**emporário (RT). Se não for possível a sua construção, a referida característica retorna como pendente ao **M**odelo **R**efinado de **C**aracterísticas, para ser refinada na próxima iteração da Engenharia Reversa. Dependendo do grau de interdependência dessa com as outras características da lista, a fase de Engenharia Avante pode ser interrompida precocemente, forçando o reinício do processo, com uma nova iteração da Engenharia Reversa. Nela, a **A**rquitetura de **R**eferência e o **M**odelo **R**efinado de **C**aracterísticas são refinados para a solução do problema.

Ao final das atividades Mineração/Refatoração, Construção de Componente-Característica e Construção de *Gateway*, a atividade Reconstrução da Aplicação recria o sistema a partir dos três repositórios existentes, *i.e.*, **R**epositório **L**egado, **R**epositório **C**ompartilhado e **R**epositório **T**emporário, sem prejuízo de nenhuma de suas características originais, produzindo uma aplicação compatível e parcialmente modernizada. Essa atividade pode ainda, por meio de configurações das variabilidades implementadas, criar novas aplicações similares, membros de uma família de produtos. A atividade Realização de Testes de Equivalência deve usar técnicas apropriadas, que não estão no escopo deste trabalho, para

aprovar a liberação da aplicação criada. Uma vez aprovada, ocorre a Atualização de Documentos, que visa reunir e manter, de forma organizada e acessível, as informações geradas durante iteração atual da Engenharia Avante. A documentação incremental produzida retrata as melhorias realizadas no código legado e também no núcleo de componentes reutilizáveis, facilitando seu reúso em aplicações similares distintas. As atividades descritas se repetem até que todas as características da lista tenham sido tratadas, quando então a iteração termina e o processo de reengenharia inicia uma nova iteração da Engenharia Reversa.

3.4 Considerações Finais

Para o domínio de sistemas embutidos, o apoio às variabilidades do hardware permite a oferta simultânea de diferentes produtos similares ao mercado, constituindo-se em uma importante estratégia de negócios para as empresas. A necessidade de criar continuamente novos produtos sugere a realização de processos mais ágeis de desenvolvimento de software embutido, adaptados a mudanças não planejadas de requisitos. Porém, manutenções freqüentes no nível de código, geralmente, aceleram a degradação das aplicações, assim como a de outros artefatos do sistema, por exemplo, a documentação, reduzindo progressivamente a agilidade desejada.

Neste capítulo foi apresentada um Processo de Reengenharia Iterativa Orientada a Características que reúne princípios ágeis e técnicas de reúso em nível mais alto de abstração para revitalizar sistemas embutidos legados, modernizando seu código, explorando o reúso de suas similaridades, melhorando o apoio a novas variabilidades e recuperando, incrementalmente, a sua documentação. A concepção desse processo apoiou-se em sugestões de outros autores em relação às melhores práticas para a solução do problema relatado e nos resultados de seus respectivos trabalhos, mencionados na Seção 3.2. A proposta resultante é uma combinação de princípios, técnicas e tecnologias, adaptados para o objetivo particular deste trabalho, presente na Seção 1.2.

O processo completo de reengenharia mostrado na Seção 3.3, não vincula suas atividades, por exemplo, testes, documentação etc., a técnicas específicas para suas realizações. Assim, o que ele oferece é um arcabouço (*framework*) de atividades que podem ser instanciadas para apoiar diferentes situações e ambientes. Como exemplo, a realização do ciclo iterativo para o desenvolvimento de componentes-características, Figura 18, pode ser

apoiado por diferentes métodos ágeis de desenvolvimento ou por uma combinação de suas práticas e artefatos, como o uso de listas de características, FDD (*Feature Driven Development*) (PALMER; FELSING, 2002), e a realização de refatorações, XP *(Extreme Programming)* (BECK, 2004). Dessa forma, a eficiência do processo pode estar vinculada à escolha apropriada de técnicas que melhor se adaptem ao ambiente, aos recursos técnicos e humanos disponíveis e ao perfil da equipe de profissionais envolvidos.

O Processo de Reengenharia Iterativa Orientada a Características proposto apresenta as seguintes vantagens em relação à proposta de Kang et al. (2005) para a criação de produtos similares, membros de uma LPS, a partir de códigos legados:

- Revitalizações concomitantes de sistemas legados similares;
- Processo ágil, com resultados rápidos, mesmo envolvendo tarefas potencialmente complexas;
- Menor risco, com avaliação precoce de custos e eficiência;
- Menor custo de produção, com fases curtas e alternadas de investimento e retorno;
- Reúso da base legada e maior simplicidade de implantação em conformidade com os princípios de aceitabilidade, descritos na Seção 3.1;
- Teste imediato dos componentes produzidos, em ambiente real;
- Apoio a variabilidades não planejadas.

Por outro lado, as desvantagens observadas são as seguintes:

- A arquitetura e o código não são integralmente modernizados;
- Manutenções não ocorrem em nível arquitetural;
- A falta de ferramentas de apoio dificulta a configuração de um número elevado de variabilidades;

Ao se comparar o processo proposto com a técnica de Mehta e Heineman (2002) os seguintes benefícios podem ser listados: 1) Além de modernizar o código legado é possível revitalizá-lo, estendendo sua funcionalidade para atender novos requisitos; 2) A criação de componentes-características reutilizáveis permite o reúso de características do software legado em outras aplicações e também a agregação de variabilidades do domínio ao seu código, com a rápida criação de uma família de produtos. 3) A substituição dos testes de regressão por modelos de características (KANG et al., 1990), permite visualizar as similaridades entre diferentes aplicações e as variabilidades do domínio, possibilitando a

criação de artefatos mais flexíveis, para apoiar um número maior de soluções; 4) O uso de *gateways* para isolar o código dos componentes do código legado (O'BRIEN, 2005) permite o uso de paradigmas mais modernos de desenvolvimento para a sua criação e também torna o processo de revitalização imperceptível para os mantenedores do código legado, que podem efetuar manutenções em pleno andamento da revitalização.

O próximo capítulo mostra o estudo de caso realizado, que utiliza o Processo de Reengenharia Iterativa Orientada a Características, aqui descrito, permitindo a avaliação de sua aplicabilidade em um domínio específico.

Capítulo 4

Estudo de Caso

4.1 Considerações Iniciais

Segundo dados da Associação Brasileira das Empresas de cartões de Crédito e Serviços, ABECS[7], consolidados até maio/2007, o Brasil possui um volume total de 389 milhões de cartões de pagamento em circulação, que movimentaram R$111,8 bilhões nesses cinco meses.

Este estudo de caso foi realizado na empresa VeriFone do Brasil Ltda (VERIFONE BR), fabricante de sistemas embutidos para o domínio de cartões de pagamento, para avaliar a aplicabilidade do Processo de Reengenharia Iterativa Orientada a Características, proposta neste trabalho. O objetivo da empresa é evoluir rapidamente e concomitantemente cada uma de suas aplicações legadas para diferentes tipos de hardware, porém similares, mantendo as regras de negócio nelas embutidas, uma vez que já passaram por um rigoroso processo de certificação e não devem ser modificadas. O código legado deve ser parcialmente modernizado e a documentação existente recuperada, facilitando manutenções futuras e adições de novos requisitos. As atividades de reengenharia devem ser imperceptíveis para os mantenedores das aplicações, que não devem ser impedidos de realizar manutenções corretivas durante o andamento do processo. As estruturas das aplicações originais não devem ser alteradas, para não dificultar o trabalho desses profissionais. A empresa espera ainda pulverizar o conhecimento de novas técnicas da engenharia de software entre seus profissionais e engajar-se em um processo contínuo de modernização e expansão da sua base de artefatos reutilizáveis, ampliando a oferta e agilizando a entrega de novas soluções aos seus clientes.

Algumas das características do ambiente de desenvolvimento da empresa são: desenvolvimento *ad hoc* realizado por um pequeno número de programadores com bom conhecimento do domínio, existência de aplicações similares distintas com baixo nível de

[7] www.abecs.org.br (09.07.07]

reúso de artefatos, bases de dados simples e pequenas e documentação de requisitos detalhada.

4.2 O Domínio de Aplicações para Terminais POS

Terminais POS (*Point Of Sale*) são equipamentos pequenos, leves, portáteis e versáteis, com um conjunto reduzido de componentes periféricos padronizados, controlado por um sistema embutido microprocessado, que atendem às necessidades de um amplo segmento de mercado. No entanto, o custo, a capacidade de processamento e armazenamento de informações, a interface simplificada com o usuário e os dispositivos de comunicação, segurança e leitura de cartões com chip e trilha magnética direcionam seu uso para a realização de Transferências Eletrônicas de Fundos (TEF) com cartões de pagamento. A Figura 19 mostra um terminal POS e seus principais componentes periféricos.

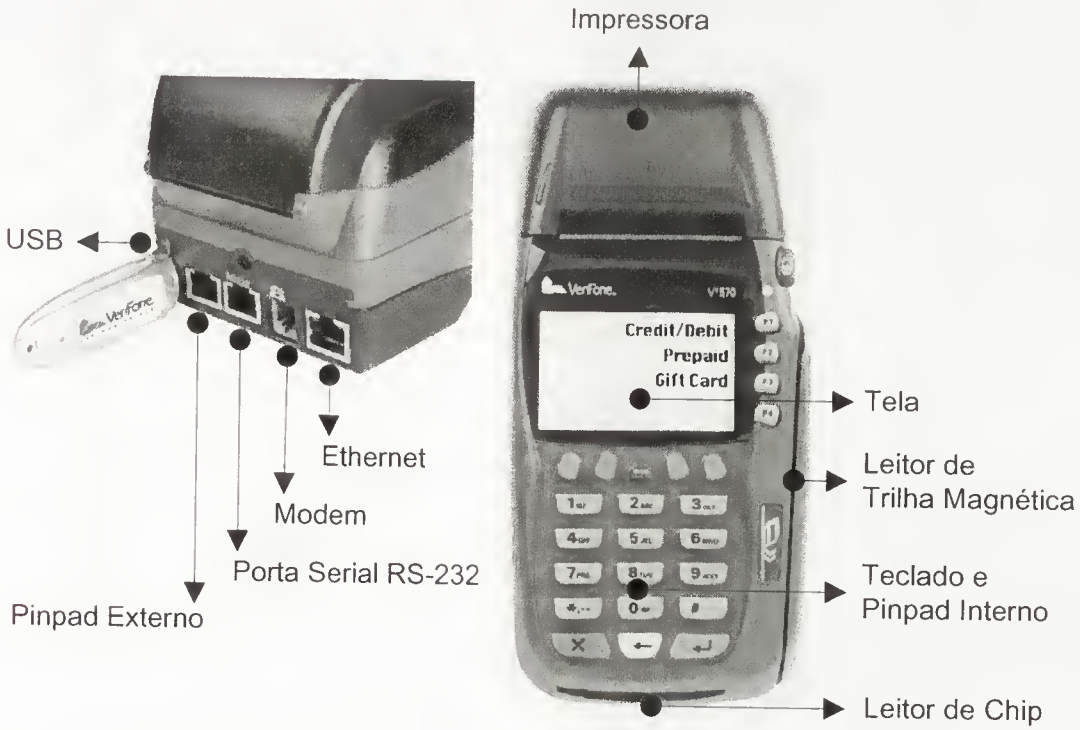


Figura 19. Terminal POS e seus principais componentes periféricos[8]

O modelo de processo para o desenvolvimento de aplicações para TEF no Brasil pode ser representado de forma genérica pelo modelo de atividades da UML (BOOCH;

[8] Terminal POS Vx-570 da empresa VeriFone Inc. (http://www.verifone.com) (09.07.07)

RUMBAUGH; JACOBSON, 1999), mostrado na Figura 20. A atividade de Elicitação de Requisitos é realizada pelo *Acquirer*[9], sem a participação ativa dos Fornecedores de POS ou dos Lojistas[10], e seu resultado é um documento de requisitos detalhado, que contém todas as informações necessárias para o desenvolvimento da aplicação. Para esse desenvolvimento o *Acquirer* não exige processos ou tecnologias específicos, que podem ser livremente adotados por cada Fornecedor de POS. Estes, por sua vez, ao receberem o documento de requisitos, iniciam o desenvolvimento de suas versões particulares da aplicação. Não há entregas parciais ao *Acquirer*. Versões incrementais, quando existentes, são testadas apenas internamente pelos Fornecedores de POS que decidiram por criá-las.

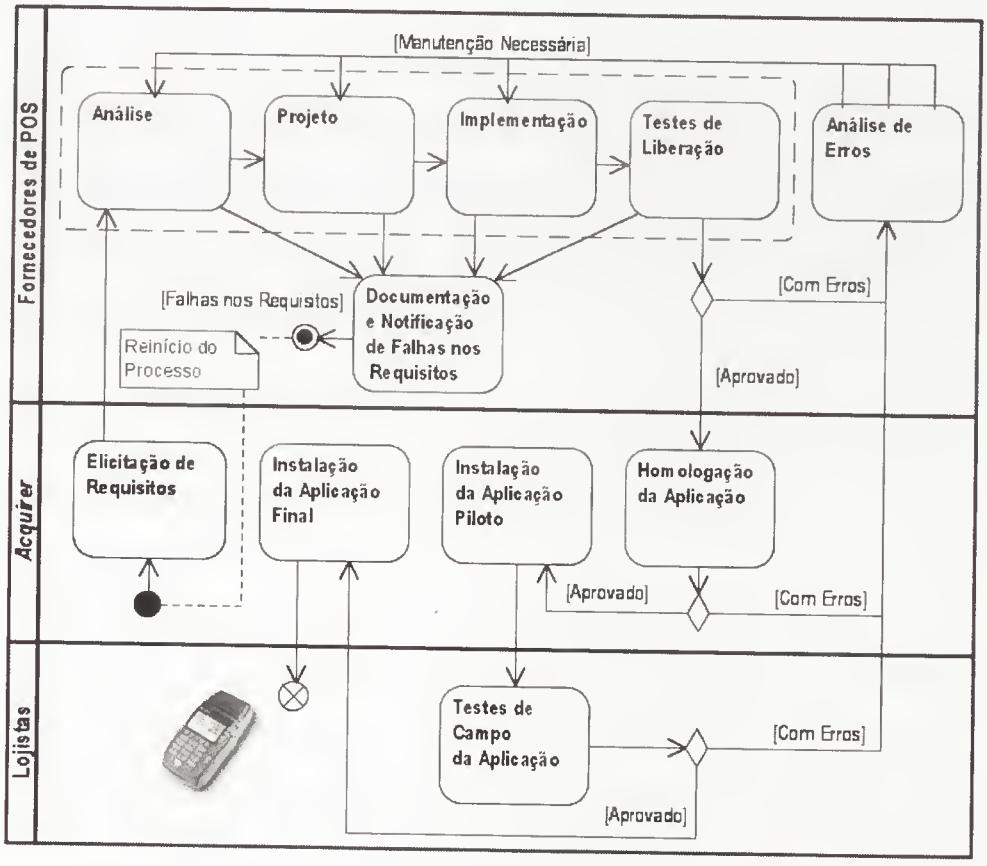


Figura 20. Modelo de processo para o desenvolvimento de aplicações para TEF no Brasil

O desenvolvimento da aplicação pode ser temporariamente interrompido caso se revelem falhas no documento de requisitos, que são documentadas e notificadas ao *Acquirer*, reiniciando o processo, como indicado pela Nota UML (BOOCH; RUMBAUGH;

[9] Banco ou empresa que processa transações com cartão de pagamento em favor dos lojistas associados
[10] Nome genérico para o usuário final das aplicações para TEF embutidas em terminais POS.

JACOBSON, 1999) presente no modelo de atividades da Figura 20. O *Acquirer* realiza as correções necessárias e o documento de requisitos, revisado, é entregue novamente aos Fornecedores de POS, que retomam o desenvolvimento interrompido anteriormente. A versão final, após ser aprovada pelos Testes de Liberação, é entregue ao *Acquirer*, que inicia a Homologação da Aplicação. Se forem encontradas falhas na aplicação, um relatório de erros é entregue ao Fornecedor de POS, que os analisa e realiza as devidas correções. Em geral, esse ciclo não pode se repetir indefinidamente sem custo para o Fornecedor de POS, dado o custo da infra-estrutura de homologação mantida pelo *Acquirer*. Após a homologação, a aplicação, denominada piloto, segue para testes de campo, geralmente inserida em terminais POS instalados em estabelecimentos comerciais específicos, que realizam um número elevado de transações diárias. A partir de um prazo estabelecido pelo *Acquirer*, se não forem detectadas falhas operacionais, a aplicação é denominada final e está apta para entrar em operação. Caso contrário, um relatório de erros, elaborado a partir dos resultados dos testes de campo, é entregue ao Fornecedor de POS, que deve providenciar as devidas correções.

4.3 Reengenharia de Sistemas Embutidos para Terminais POS

Para a realização deste estudo de caso, foram tomadas três aplicações legadas similares para TEF, desenvolvidas para terminais POS, pertencentes à empresa VeriFone do Brasil, responsável por gerir o sigilo das informações contidas nos artefatos disponibilizados. Os documentos de requisitos dessas aplicações, juntamente com seus códigos legados, foram entregues para o autor desta dissertação. As aplicações são distintas, sem documentação de projeto, possuem estrutura modular e código escrito em linguagem C com níveis insatisfatórios de reúso, apoiado apenas por agrupamentos de funções genéricas em módulos compartilhados de linkedição (*libraries*). Participaram deste estudo de caso o autor desta dissertação, um profissional de testes e dois programadores da empresa.. Durante a reengenharia das aplicações esses programadores atuaram como Especialistas do Domínio.

A seguir são descritas as atividades realizadas e apresentados os artefatos produzidos pelo Processo de Reengenharia Iterativa Orientada a Características, no qual foi realizada a manutenção adaptativa dos sistemas legados, a modernização de seus códigos, a recuperação gradativa de sua documentação e a produção de artefatos reutilizáveis.

A primeira iteração da Engenharia Reversa teve início com uma reunião envolvendo áreas estratégicas da empresa, na qual foi feito o levantamento das deficiências e das necessidades imediatas e futuras das três aplicações, na visão de cada participante. De posse dessas informações e dos artefatos fornecidos, foi possível efetuar uma análise preliminar de viabilidade técnica do projeto de reengenharia das aplicações. Diante do interesse em preservar as regras de negócio e permitir variações do hardware, decidiu-se pela revitalização gradativa e concomitante das aplicações, com enfoque nas características do hardware. Para facilitar a validação preliminar do processo, a partir de seus primeiros resultados, foram priorizadas as variabilidades do hardware de baixo impacto operacional. Definida a estratégia, o domínio foi estudado com o apoio dos **E**specialistas do **D**omínio designados pela empresa. Após a leitura da documentação existente, os códigos legados foram analisados e as dúvidas esclarecidas para entendimento dos elementos do hardware e seu uso pelas aplicações.

Com essas informações foi realizado o Esboço da Arquitetura Legada, que resultou na representação da **A**rquitetura **B**ase, ilustrada na Figura 21. Tentativas *ad hoc* anteriores de estruturação das aplicações, por parte da empresa, produziram uma Biblioteca de Funções comum e um conjunto de Módulos Selecionáveis[11] que, por meio de compilação condicional, implementam algumas variabilidades. SDK (***S**oftware **D**evelopment **K**it*) é um pacote de desenvolvimento de software para terminais POS, fornecido pela empresa, que permite acessos aos serviços oferecidos pelo Sistema Operacional (SO).

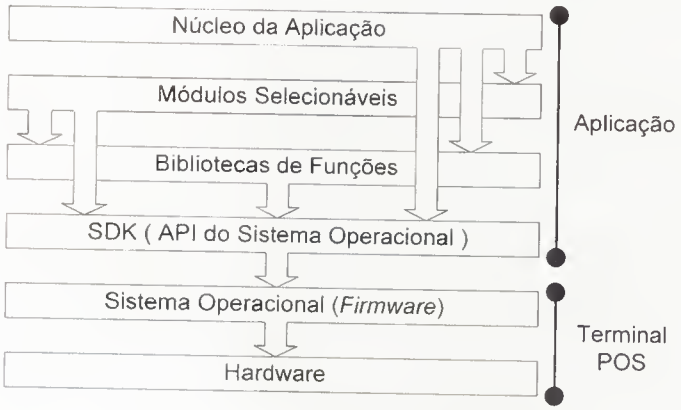


Figura 21. Arquitetura Base (AB)

[11] Denominação interna, criada pelos programadores da empresa.

O Modelo Inicial de Características, Figura 22, é resultante da atividade de Criação do Modelo de Características (KANG et al., 1990) e retrata apenas a estrutura comum das aplicações do domínio, ocultando os detalhes ainda desconhecidos, contidos nos códigos legados das aplicações. As características associadas ao hardware foram agrupadas separadamente, favorecendo seu detalhamento de forma independente, conforme solicitação da empresa.

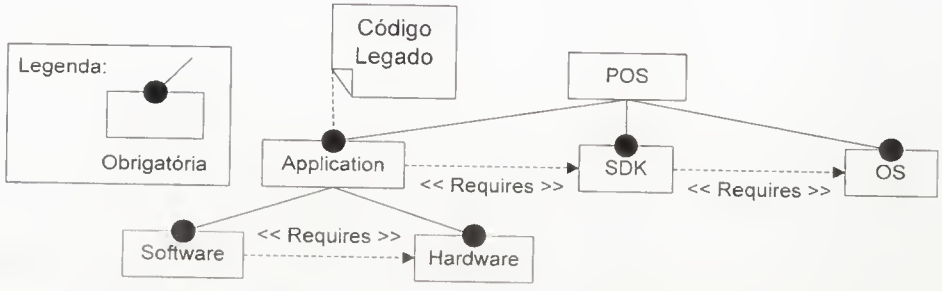


Figura 22. Modelo Inicial de Características (MIC) [12]

Como as aplicações legadas não implementam apoio à conexão de teclado externo e sendo essa uma variabilidade de baixo impacto operacional e importante para a estratégia de negócios da empresa, optou-se por validar o processo a partir de sua implementação. Ao realizar a atividade Refinamento do Modelo de Características o primeiro Modelo Refinado de Características foi produzido a partir do Modelo Inicial de Características e retrata essa variabilidade, Figura 23. Para a modelagem correta do domínio, o Modelo Inicial de Características foi mantido como um subconjunto do Modelo Refinado de Características, ilustrando a coexistência do código legado e do modernizado. O *Gateway*, por não ser uma característica do domínio, mas sim uma técnica de implementação, foi representado como uma relação de uso[13] do código legado para com código compartilhado.

Uma decomposição estrutural (OLSEM, 1998) foi realizada para retratar os elementos do hardware e os seus relacionamentos em diferentes níveis de abstração e para facilitar o seu mapeamento posterior em classes de componentes. Por exemplo, `IO` representa o conjunto de periféricos que compõe o hardware ao lado de outros elementos como memória e microprocessador. `Device`'s são *`IO`*'s mapeados em portas de Entrada/Saída (E/S). `ComPort`'s

[12] Os nomes das características estão em inglês para criar uma correspondência direta com a nomenclatura usada na implementação de seus códigos.

[13] Relação representada pelo estereotipo <<uses>>. na qual um participante pode opcionalmente utilizar recursos do outro, mas não depende desse para sua existência..

são Device's configuráveis dedicados à troca de dados com outros dispositivos. Devido à presença do SDK, características físicas e lógicas do hardware foram representadas no **M**odelo **R**efinado de **C**aracterísticas para representar não só a constituição física do equipamento, mas também a sua estruturação lógica do ponto de vista do sistema operacional. Como resultado pode-se observar, por exemplo, a Console (característica lógica) constituída de um ou mais teclados, Internal e External Keyboard, (características físicas). Essa representação também visa a facilitar o mapeamento característica-componente.

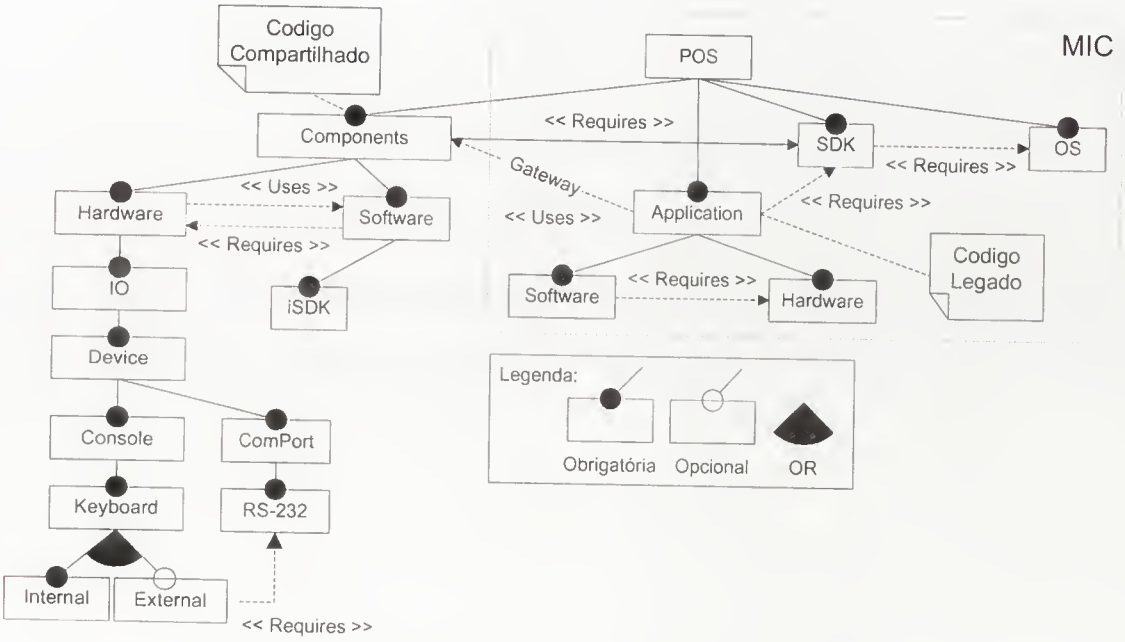


Figura 23. Modelo Refinado de Características (MRC)

O teclado externo foi representado como uma característica opcional e quando conectado à característica RS-232 deve operar em conjunto com o teclado interno obrigatório (OR).

A partir do **M**odelo **R**efinado de **C**aracterísticas e da **A**rquitetura **B**ase foi realizada a Transformação da Arquitetura originando a primeira **A**rquitetura de **R**eferência, Figura 24.

As aplicações legadas compartilham o uso dos artefatos contidos no núcleo de componentes de forma imperceptível, por intermédio do *Gateway*, que atua como um adaptador de interface. O núcleo da aplicação foi isolado do *Gateway*, deixando as regras de negócio separadas do processo de reengenharia, conforme requisito descrito anteriormente. O iSDK é um elemento de interface que permite isolar os componentes das variações do SDK, aumentando sua portabilidade. A criação da **A**rquitetura de **R**eferência encerrou a primeira

iteração da Engenharia Reversa do Processo de Reengenharia Iterativa Orientado a Características. Em seguida, teve início a primeira iteração da Engenharia Avante para a implementação dos primeiros artefatos reutilizáveis do núcleo de componentes e a reconstrução das aplicações legadas, parcialmente modernizadas.

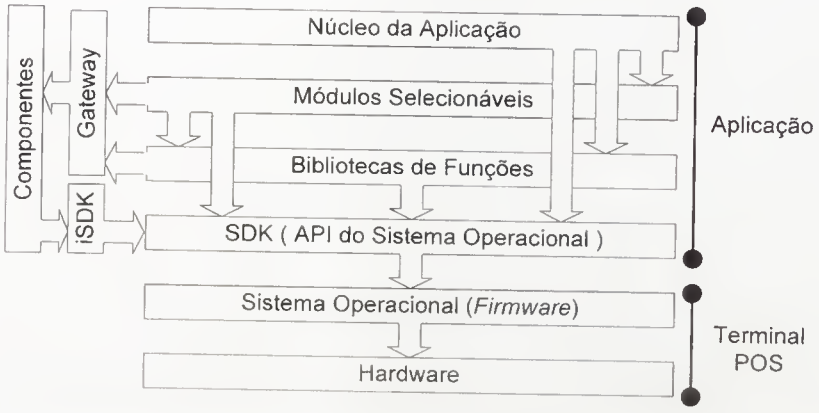


Figura 24. Arquitetura de Referência (AR)

A primeira Lista de Características foi elaborada com base no Modelo Refinado de Características, contendo as características `Internal` e `External Keyboard`, todas que as precedem em seus ramos de árvore (dependência vertical) *i.e.*, `IO`, `Device`, `Console` e `Keyboard`, e todas as outras com as quais mantém uma relação de dependência direta ou indireta (dependência horizontal), *i.e.*, `ComPort` e `RS-232`.

Ao selecionar a primeira característica, *i.e.*, `IO`, observou-se um problema com relação à formação e representação de características no Modelo Refinado de Características, diferente de Divisão e Junção, identificadas por Sochos, Philippow e Riebisch (2004), denominado "Pulverização", que está ilustrado na Figura 25. Nesse caso, uma característica é decomposta excessivamente, criando sub-características não visíveis ao usuário, mas que atuam como elementos estruturais para a formação de outras características representativas do domínio, ajudando a melhorar a compreensão dessas. Em termos de mapeamento característica-componente representam classes abstratas formadoras de um componente, que isoladamente não representam uma propriedade do sistema ou ampliam sua funcionalidade. Em atividades de mineração, representam características que não podem ser extraídas isoladamente, mas que integram a extração da característica visível a qual pertencem.

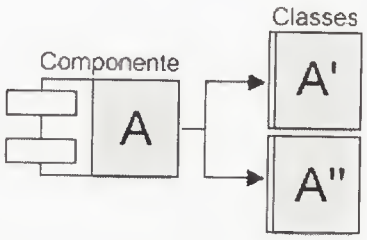


Figura 25. Pulverização de Características

Para representar sub-características em um modelo de características, foram criadas duas novas extensões para o modelo proposto por Kang et al. 1990, descritas a seguir.

- **Nota**: É representada como na UML (BOOCH, RUMBAUGH E JACOBSON, 1999), podendo estar conectada a uma ou mais características por meio de âncoras. Por questão de legibilidade, possui internamente apenas um número seqüencial, que faz referência a um texto explicativo externo ao modelo. Ela agrega ao modelo de características detalhes do domínio que não podem ser representados graficamente e que desempenham um papel importante na tomada de decisões de projeto. Pode ainda conter informações temporárias, em um determinado nível de abstração, que servirão como guias para refinamentos do modelo em etapas posteriores do projeto. Quando o texto da Nota descreve uma informação relevante para a implementação da característica, uma letra '**i**' é acrescentada à frente do número seqüencial
- **Sub-Característica**: Representada por um contorno pontilhado, indicando que é uma característica não visível ao usuário, mas importante como elemento estrutural dos produtos desse domínio.

A partir dessas considerações a estrutura da Lista de Características foi alterada para permitir uma representação distinta das características visíveis e suas sub-características, como mostra a Figura 26.

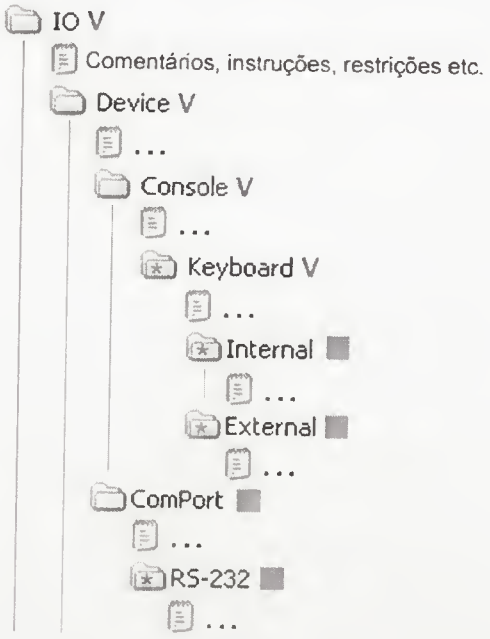
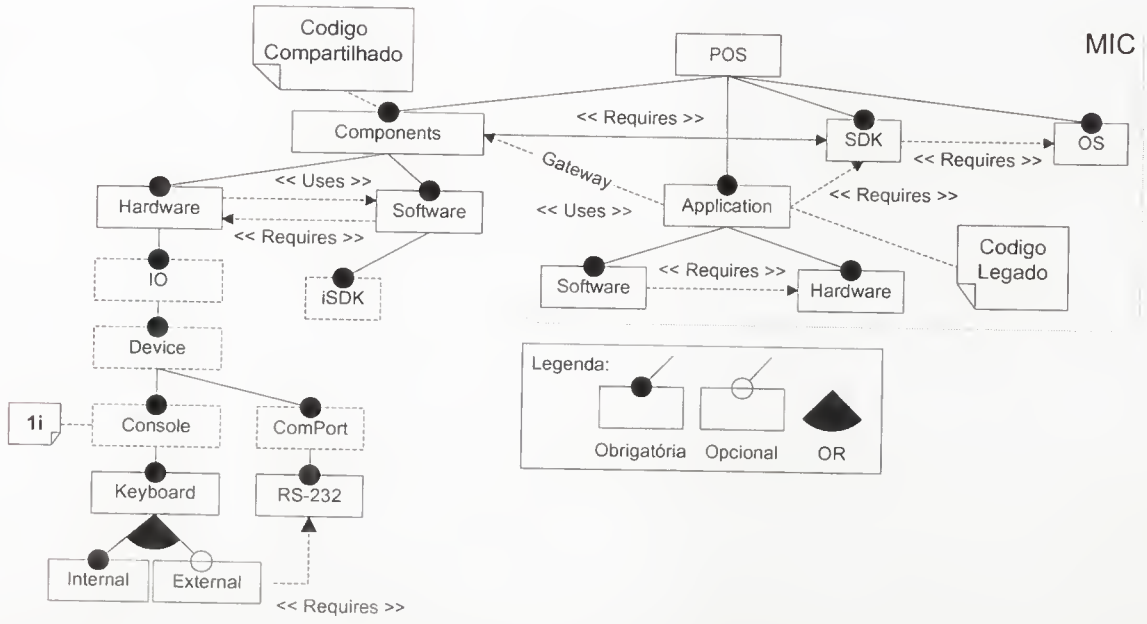


Figura 26. Estrutura da Lista de Características modificada

A forma adotada foi a de uma lista hierárquica que permite: a) distinguir sub-características de características visíveis, assinaladas com '*', b) visualizar suas hierarquias c) adicionar comentários individuais e relevantes para suas implementações e d) marcar as já tratadas, com o símbolo *V*.

O **M**odelo **R**efinado de **C**aracterísticas também foi modificado para incluir as extensões criadas, como mostra a Figura 27.

Ao final das atualizações e com a ajuda de programadores da empresa, foram realizadas inspeções nos códigos legados das três aplicações para identificar as funções diretamente associadas ao teclado, iniciando a construção do Mapa de Conexões da característica Keyboard. Para apoiar o desenvolvimento do seu componente-característica foram captadas, adicionalmente, informações sobre as propriedades desejáveis e as limitações existentes, que foram documentadas no Mapa de Conexões mostrado na Tabela 2.



1i O SO redireciona as leituras do Console para o Teclado e as escritas para a Tela do POS.

Figura 27. MRC modificado com as extensões Nota e Sub-Característica

Devido ao impacto dos novos requisitos obtidos na criação do Mapa de Conexões para a característica `Keyboard`, a iteração atual de Engenharia Avante foi interrompida, dando início a uma nova iteração da Engenharia Reversa para a revisão de seus artefatos, *i.e.*, Arquitetura de Referência e Modelo Refinado de Características.

Tabela 2. Mapa de Conexões (MCo) da Característica Keyboard

Característica:		**IO Device Console Keyboard**			
Função	**Grupo**	**Parâmetros**	**Retorno**	**Comentários**	**Aplicações**
KBHIT	1	Nenhum	Status: bool	Retorna TRUE se houver tecla pressionada.	#1, #2, #3
get_char	2	Nenhum	Tecla : int	Aguarda o pressionamento de uma tecla e retorna seu valor	#1, #2, #3

[Limitações]

- O sistema operacional não permite escritas no *buffer* do teclado interno;
- O teclado externo não possui um buffer de teclas provido pelo sistema operacional.

[Novos Requisitos]

- Os teclados, interno e externo, devem compartilhar um *buffer* comum de teclas, provido pelo componente, o qual deve permitir operações de escrita e leitura;
- O componente deve incluir um controle dinâmico de liga/desliga para os teclados ativos;
- A fim de flexibilizar o seu uso, a conexão do teclado externo pode ser feita nas portas RS-232 ou *PinPad* do terminal POS, com parâmetros de comunicação configuráveis externamente à aplicação.

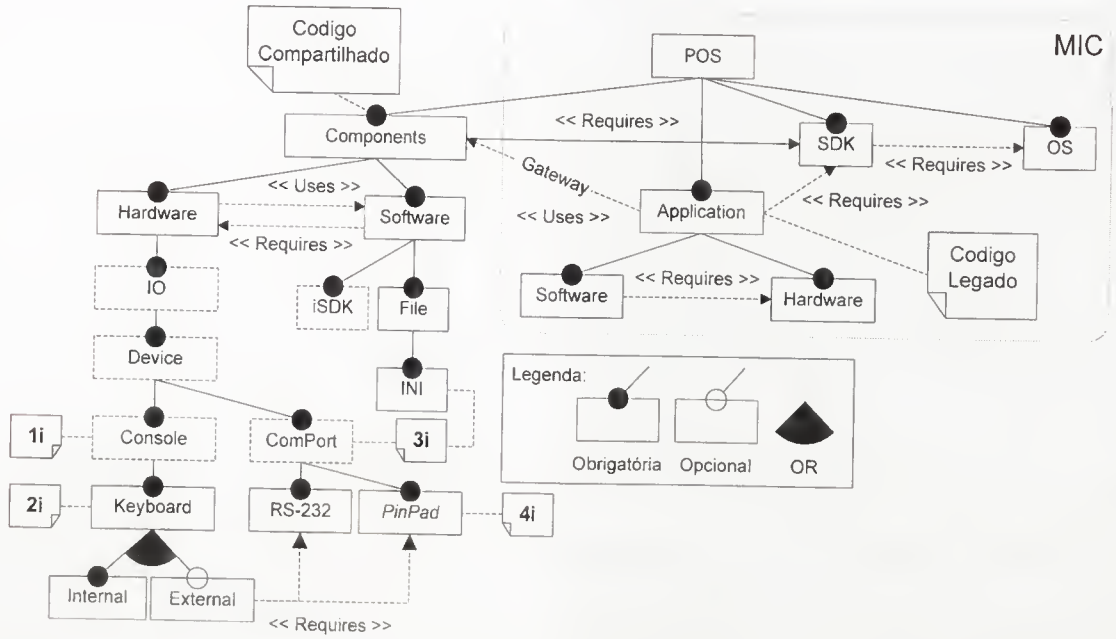
O resultado foi o Modelo Refinado de Características mostrado na Figura 28. Em relação ao modelo anterior, foram adicionados uma dependência do teclado externo em

relação à característica `PinPad` e uma nova característica de software, inexistente no código legado, os arquivos tipo `INI`[14], para permitir o armazenamento dos parâmetros de comunicação externamente à aplicação.

Retomada a Engenharia Avante, a atividade de Construção de Componente-Característica foi iniciada com o mapeamento do **M**odelo **R**efinado de **C**aracterísticas, Figura 28, para um modelo de classes em UML (BOOCH; RUMBAUGH; JACOBSON, 1999), apoiado pelo **M**apa de **Co**nexões e pela documentação do sistema operacional, que contêm os serviços disponibilizados para cada dispositivo do hardware.

O resultado é o modelo de classes mostrado na Figura 29, no qual os teclados interno, `IKeyboard`, e externo, `EKeyboard`, foram unificados pela classe única `Keyboard`. Essa classe atua como gerenciador para ambos os teclados e cria obrigatoriamente um objeto para o teclado interno e opcionalmente outro para o teclado externo, dependendo da configuração dessa variabilidade. Um *buffer* circular interno armazena para leitura os códigos das teclas pressionadas em qualquer um dos teclados ativos e, adicionalmente, permite a escrita de códigos de teclas, simulando ações reais dos teclados. Variáveis internas fazem o controle individual da habilitação dos teclados ativos, permitindo ligá-los e desligá-los dinamicamente durante o andamento da aplicação. A classe associada ao teclado externo, `EKeyboard`, usa polimorfismo para criar um objeto específico da classe `ComPort`, a partir das classes RS-232 ou `Pinpad`, e, por meio desse, realizar a comunicação serial com o teclado.

[14] Arquivo padrão do sistema operacional Windows para o armazenamento de configurações do ambiente.



1i — O SO redirecciona as leituras do Console para o Teclado e as escritas para a Tela do POS.

2i — Deve permitir escritas e leituras

3i — Deve permitir a configuração de parâmetros externamente à aplicação

4i — A porta física de E/S para *PinPad* Externo e Interno é única (compartilhada), mas pode possuir endereços lógicos diferentes (e.g. COM2, COM5, etc). Quando os endereços lógicos coincidem, é necessário um procedimento especial para a seleção correta do dispositivo de E/S desejado. Caso contrário, essa seleção é feita automaticamente pelo S.O. com base no endereço lógico utilizado.

Figura 28. MRC refinado para acomodar os novos requisitos contidos no MCo.

À classe `ComPort` foi adicionado um método para a leitura de seus parâmetros de comunicação, contidos em arquivos do tipo `INI`, externos à aplicação. Esse tipo de arquivo foi escolhido por atender à necessidade de armazenamento das configurações do hardware e pela maior facilidade de implementação de funções para a manipulação de seus dados. Essas implementações atendem a todos os requisitos originais e também os novos, contidos no **M**apa de **Co**nexões da característica `Keyboard`, Tabela 2.

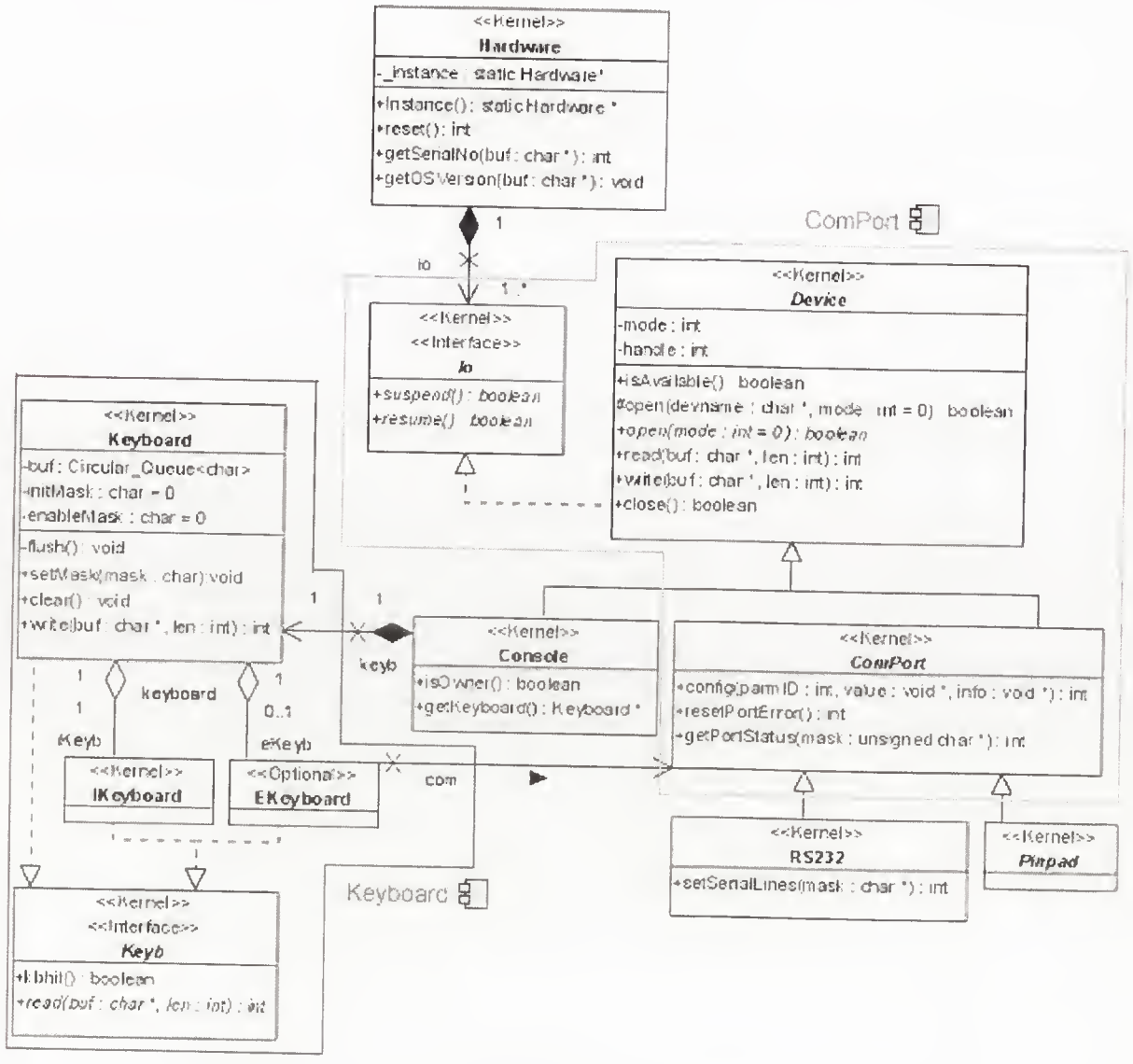


Figura 29. Modelo de Classes da Característica `Keyboard`

A classe para tratamento de arquivos do tipo `INI`, Figura 30, implementa a busca e leitura de sessões do arquivo, delimitadas por colchetes, Figura 32, e armazena os valores de suas chaves de configuração na memória, permitindo consultas posteriores.

A partir desses modelos foi desenvolvido um modelo de componentes para documentar os serviços oferecidos pelos novos artefatos reutilizáveis criados, como mostra, de forma simplificada, a Figura 31.

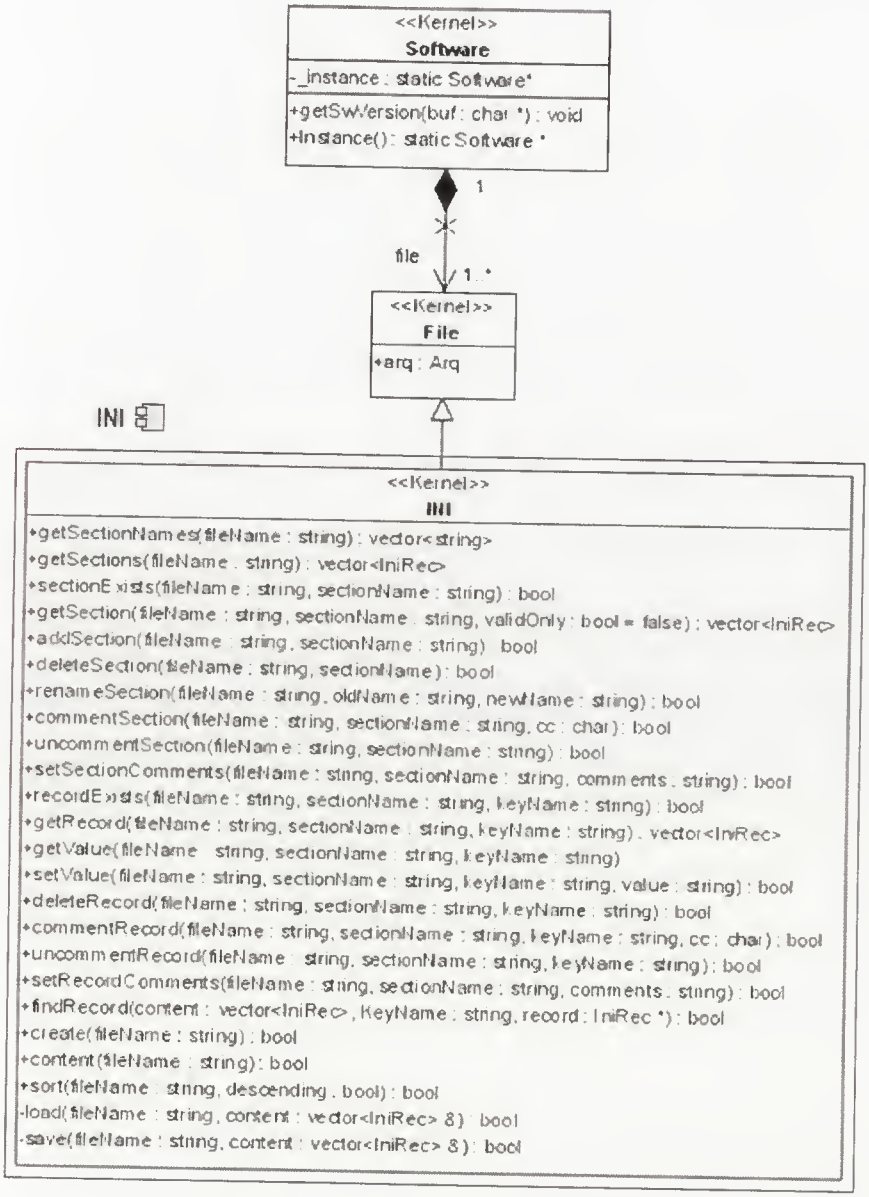


Figura 30. Modelo de Classes da Característica `INI`

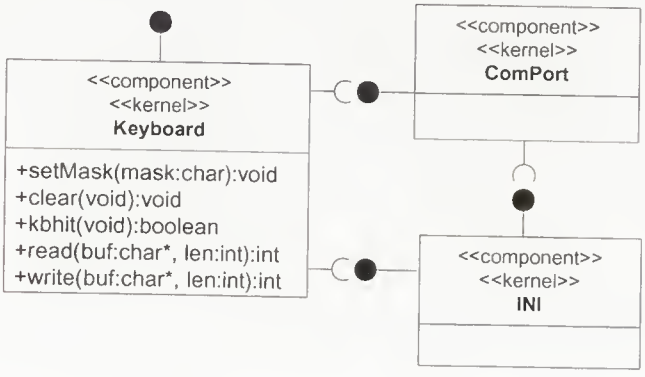


Figura 31. Modelo simplificado de componentes da característica `Keyboard`

A Figura 32 mostra o arquivo de configuração do tipo INI criado, com as duas possibilidades de conexão para o teclado externo, em conformidade com os requisitos dessa característica. Esse arquivo pode ser o princípio para a criação de uma LMA (**L**inguagem de **M**odelagem de **A**plicações), capaz de agrupar seletivamente as variabilidades de um domínio para a produção de uma linha de produtos sem re-projeto ou re-codificação das aplicações.

```
[CONSOLE]
KEYBOARD=EXT-PINPAD
[EXT-RS232]
# <1 = COM1, 2 = COM2>
COM=1
# MODE: <protocol>, <baudrate>, <data length>, <parity>, <stop bits>, <stx_char>, <etx_char>
MODE=CHAR,115200,8,N,1
[EXT-PINPAD]
COM=2
MODE=CHAR,19200,7,E,1
```

Figura 32. Arquivo do tipo INI para Configuração do Teclado Externo

Uma vez criados os componentes foi iniciada a implementação do *gateway* (O'BRIEN, 2005), guiado pelo conteúdo do **M**apa de **Co**nexões. Como as funções listadas representam chamadas diretas à API (*Application Programming Interface*) do sistema operacional, não houve a necessidade de remover suas respectivas implementações do código legado. Porém, devido ao conflito de nomes, essas foram renomeadas e receberam um prefixo _G_ para identificar que estão implementadas no *gateway*. Nenhuma outra alteração foi feita no código legado das aplicações. O *gateway* desenvolvido para o componente-característica `Keyboard` é mostrado na Figura 33.

Por fim, os códigos do *Gateway* e dos componentes reutilizáveis criados foram compilados com as três aplicações legadas individualmente, que passaram a admitir opcionalmente o uso de teclado externo nas portas RS-232 ou *Pinpad*, configuradas por meio de um arquivo do tipo INI, externo à aplicação.

Testes de equivalência foram realizados por funcionários da empresa, que possuem acesso a um conjunto apropriado de ferramentas, disponibilizado pelos *Acquires*. A documentação foi revisada pela equipe envolvida com a reengenharia e os resultados do trabalho foram levados ao conhecimento da empresa, que os aprovou bem como a continuidade do projeto.

```
#ifndef _GATEWAY_C_
#define _GATEWAY_C_

#include <string>
using namespace std;

#include "Console.h"
#include "Keyboard.h"
#include "Sdk.h"
#include "Gateway.h"

namespace Hw
{    class Console   ;
     class Keyboard  ;
}
// Keyboard variability control through configuration file

Hw::Console*     _c = NULL ;
Hw::Keyboard*    _k = NULL ;

void initConsole(void)
{
          _c = new Hw::Console("HARDWARE.INI");
     _k = _c->getKeyboard();
}
int _G_kbhit (void)
{        if   (    !_k )initConsole( ) ;
     return  (int)_k->kbhit( )        ;
}
int _G_getchar(void)
{
     char     ch                      ;
     while(!_G_kbhit())               ;
                  _k->read(&ch, 1)    ;
     return  (int)     ch             ;
}

#endif/*_GATEWAY_C_*/
```

Figura 33. *Gateway* para o componente-característica `Keyboard`

4.4 Discussão sobre os Resultados

A abordagem de reengenharia iterativa orientada a características proposta foi aplicada neste estudo de caso para a revitalização concomitante de três sistemas legados para TEF no domínio cartões de pagamento, embutidos em terminais POS. Sua realização resultou em códigos parcialmente modernizados, armazenados em um Repositório Legado, em componentes-características reutilizáveis, armazenados em um Repositório Compartilhado e em adaptadores de interface, armazenados em um Repositório Temporário. As aplicações

resultantes da união de códigos desses três repositórios são compatíveis com suas versões originais, porém agregam novas funções, que foram exploradas e resultaram na criação de aplicações similares, membros de uma família de produtos, que apóiam as variabilidades do hardware. Assim, os resultados obtidos satisfazem o objetivo estabelecido inicialmente pelo cliente de evoluir rapidamente e concomitantemente cada uma de suas aplicações legadas para diferentes hardwares, porém similares, mantendo intactas as regras de negócio nelas embutidas.

Em conformidade com o resultado previsto por Crnkovic (2005), Seção 1.1, os componentes de software embutido foram criados como extensões de código para as aplicações e unidos a ela durante a sua criação (*build-time*). A Figura 34, mostra a estrutura de arquivos criada para armazenar o código dos componentes reutilizáveis, *i.e.*, Repositório Compartilhado, e o arquivo VxSPL.a de linkedição, que agrupa e disponibiliza para uso os componentes já desenvolvidos.

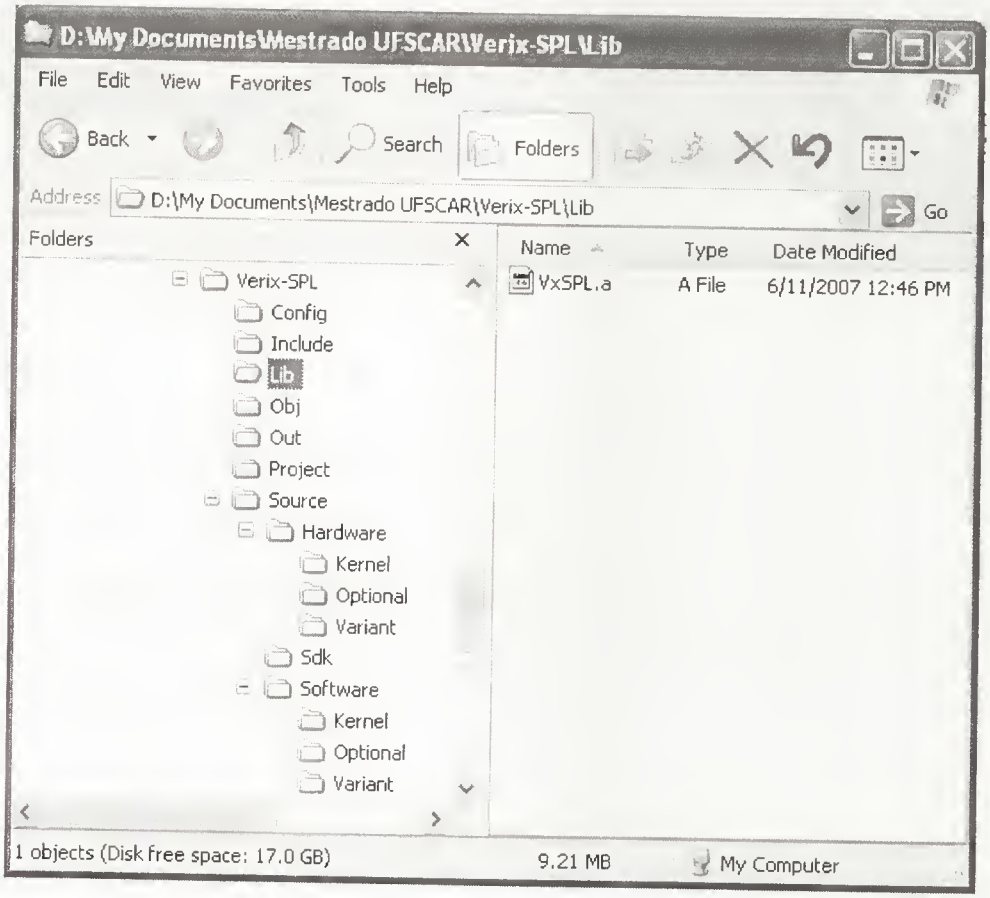


Figura 34. Estrutura de arquivos do Repositório Compartilhado

Sem um conjunto de ferramentas para criar automaticamente a aplicação, essa tarefa foi feita manualmente. Isso difere da forma de criar produtos, prevista para a Engenharia de Aplicação, em uma Linha de Produtos de Software. O resultado, porém, foi satisfatório devido ao baixo número de variabilidades envolvidas.

O código do *Gateway*, Figura 33, foi inserido no processo de criação da aplicação, como mostra a Figura 35.

```
#       ------------------------------------------------------------------------
#       File            :       V30Prd.cmp
#       Date            :       Jul 09, 2007
#       Project         :       Visanet application V30
#
#       Copyright (C) 2007 by VeriFone, Inc.
#       All rights reserved.
#       ------------------------------------------------------------------------
#       Revisions:
#       Date            Name                    What
#       Jul 09, 2007    Marcelo_R1              Creation
#       ------------------------------------------------------------------------

#  Compiler/Linker/Outhdr Output Paths
ObjDir = $(VRSDIR)\Obj
LibDir = $(VRSDIR)\Lib

# \GATEWAY
BldFile200 = Gateway
SrcFile200 = GATEWAY\$(BldFile200)

# ------------------------------------------------------------
Appbanner:
    @echo.
    @echo **----------------------**
    @echo ** MAIN APPLICATION --- **
    @echo **----------------------**
    @echo ** COMPILING CODE FILES **
    @echo **----------------------**
       @echo.
       @cd $(ObjDir)
# ------------------------------------------------------------

$(ObjDir)\$(BldFile200).o : $(SRCDIR)\$(SrcFile200).cpp
  @echo $(BldFile200).cpp
  @$(SDSTOOLS)\armcc $(Includes) $(COptions) $(SRCDIR)\$(SrcFile200).cpp
```

Figura 35. Código do Gateway inserido na construção da aplicação

Por meio de inspeções no código legado da aplicação, as ocorrências das funções presentes no **M**apa de **C**onexões, Tabela 2, foram substituídas pelas funções do *Gateway*, Figura 36. A aplicação foi reconstruída com o uso dos componentes contidos no arquivo VxSPL.a, Figura 37. Por fim, a aplicação foi transferida para o terminal POS, juntamente com o arquivo de configuração Hardware.ini, Figura 32. A partir deste ponto, o terminal pôde operar com os dois teclados, interno e externo, como planejado inicialmente.

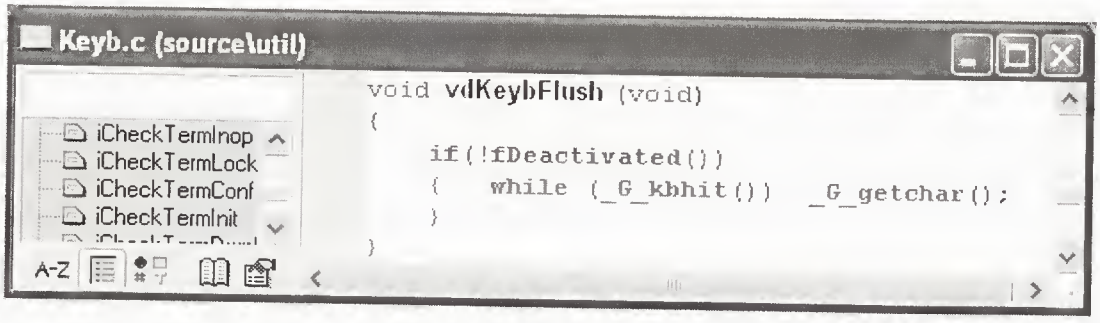


Figura 36. Substituição no código da aplicação das funções implementadas no *Gateway*

```
#       ------------------------------------------
#       File            :       V30Prd.lnk
#       Date            :       Jul 09, 2007
#       Project         :       Visanet application V30
#
#       Copyright (C) 2007 by VeriFone, Inc.
#       All rights reserved.
#       ------------------------------------------
#       Revisions:
#       Date            Name                    What
#       Jul 09, 2007    Marcelo_R1              Creation
#       ------------------------------------------
#Targets
TgtApp = $(VERSNAME)
TgtTsk =    Update

#Files to build
AppModulesLibs = \
$(AppLibraries)\$(TgtApp)1.a \
$(AppLibraries)\$(TgtApp)2.a

TskModulesLibs = \
$(AppLibraries)\$(TgtTsk).a

FoundationLibs =  \
$(VXSPLLibraries)\vxspl.a \
$(VXSTLLibraries)\vxstl.a \
$(ACTLibraries)\act2000.a \
$(SDSLibraries)\Verix.lib \
$(SDSLibraries)\Voy.lib
```

Figura 37. Reconstrução da aplicação com os componentes contidos no arquivo vxSPL.a

Semelhante à proposta do método FAST (WEISS; LAI, 1999) o arquivo Hardware.ini pode ser considerado uma LMA simples, capaz de criar variações de uma aplicação por meio de parametrizações. No entanto, essa forma de produzir aplicações similares assemelha-se mais àquela utilizada pelo método KobrA (ATKINSON et al., 2001). Assim, não há uma semelhança clara do processo de reengenharia proposto com nenhum desses dois métodos para criação de Linha de Produtos de Software, dos quais foram consideradas apenas algumas de suas propriedades, como as que acabaram de ser mencionadas.

4.5 Considerações Finais

O estudo de caso realizado permitiu o gerenciamento de variabilidades do hardware por meio de componentes de software parametrizáveis, isolados do código legado e conectados a ele por meio de *gateways*, sem alterar sua estrutura ou regras de negócio. Do ponto de vista de seus mantenedores, a reengenharia foi realizada de maneira imperceptível, sem ocasionar interferências ou paralisações de suas atividades diárias.

A introdução de práticas ágeis ao processo de reengenharia minimizou o impacto de implantação do processo e de novos paradigmas e ferramentas de desenvolvimento, que foram introduzidos incrementalmente, iniciando com a realização de tarefas menores e menos complexas. A capacidade de reação rápida a mudanças não planejadas de requisitos foi exercitada satisfatoriamente, como no caso da conexão do teclado externo em portas diferentes. A liberação de versões incrementais modernizadas das aplicações legadas facilitou as atividades de teste, que podem enfocar em uma única característica por versão. A recuperação da documentação foi feita gradativamente, permitindo uma averiguação contínua de possíveis defasagens com o código.

A reengenharia parcial planejada resultou na revitalização das três aplicações legadas disponíveis concomitantemente, com baixo risco para o cliente, que pôde avaliar precocemente os custos e a eficiência do processo, a partir de seus primeiros resultados.

Técnicas voltadas para o reúso permitiram a criação de um núcleo de artefatos reutilizáveis a partir do código legado e, a partir desse, de sistemas similares, membros de uma família de produtos.

A partir do Mapa de Conexões foi possível visualizar a conexão de uma característica com o código de diferentes aplicações. Assim, a interface de seu componente pôde ser projetada de forma genérica o suficiente para dar suporte a todas as funções captadas, permitindo a modernização de várias aplicações similares concomitantemente.

A captação contínua das necessidades do mercado, com o envolvimento de pessoas associadas direta ou indiretamente ao sistema, permitiu identificar, classificar e priorizar as mudanças necessárias, reduzindo o percentual de falhas associadas aos requisitos.

Capítulo 5

Conclusão

5.1 Contribuições

Nesta dissertação foi proposto um Processo de Reengenharia Iterativa Orientada a Características para sistemas pertencentes ao domínio de aplicações para TEF com cartões de pagamento embutidas em terminais POS. A utilização desse processo possibilita revitalizar sistemas embutidos legados de forma gradativa e com tecnologia mais atual do que aquela na qual eles foram desenvolvidos. O código legado é parcialmente modernizado e continua atendendo às necessidades da empresa.

As principais contribuições desta dissertarão são apresentadas a seguir:

- Um processo de reengenharia que permite integrar princípios ágeis e técnicas de Linha de Produtos de Software para revitalizar sistemas embutidos legados;
- A criação de uma família de produtos a partir de sistemas embutidos legados. Os produtos dessa família podem ser facilmente criados para apoiar as variabilidades possíveis do hardware que acompanha tais sistemas;
- A criação e o uso de **Ma**pas de **Co**nexão como artefatos de apoio à extração de características a partir de códigos legados;
- A exemplificação do uso de arquivos do tipo INI como uma possível LMA (WEISS; LAI, 1999), permitindo o armazenamento e a configuração das variabilidades de um domínio, apoiando a produção de uma linha de produtos sem re-projeto ou re-codificação das aplicações;
- A confecção de um núcleo de componentes reutilizáveis, possibilitando a criação de produtos similares de uma Família de Produtos de Software;
- O reúso de código legado de forma compatível com os princípios de aceitabilidade comentados por Graaf; Lormans e Toetenel (2003);
- O uso das abordagens *top-down* e *bottom-up* combinadas para a revitalização dos sistemas embutidos legados;

- A documentação de projeto dos sistemas legados pode ser parcialmente recuperada o que facilita as futuras manutenções desses sistemas.
- A criação de extensões para o modelo de características proposto por Kang et al. (1990) ampliando a sua capacidade de modelar particularidades de um domínio. Notas e Sub-características foram adicionadas ao modelo com representações particulares. Por meio das Notas permite-se a documentação de requisitos relevantes do domínio. Sub-características possibilitam a representação de características não visíveis ao usuário, porém importantes para o entendimento do domínio.
- O uso de sub-características como um elemento útil para facilitar o mapeamento característica-componente.
- O início da revitalização de sistemas embutidos legados existentes na empresa VeriFone, permitindo-lhe suporte a variações do hardware de seus produtos, adição de novas funções às aplicações existentes e a documentação de projeto que favorece manutenções futuras.

5.2 Limitações do Projeto

O processo proposto apresenta as seguintes limitações:

- As aplicações embutidas em terminais POS que foram utilizadas para realização do processo proposto, possuem particularidades que favorecem a realização das atividades de reengenharia, tais como o tamanho reduzido do código legado codificado em linguagem C, o tamanho e a simplicidade das bases de dados, a disponibilidade de documentos de requisitos detalhados e a existência de especialistas do domínio;
- O desenvolvimento do processo voltado para as necessidades de um determinado tipo de sistemas embutidos legados;
- A criação dos componentes durante o processo de reengenharia proposto não pôde ser realizada com tecnologias atuais, como EJB, Corba;
- Os componentes gerados durante o processo de reengenharia não podem ser disponibilizados de forma comercial, uma vez que estão associados a um hardware específico.

5.3 Trabalhos Futuros

A partir do processo aqui apresentado alguns trabalhos podem ser realizados para sua complementação e ampliação. Dentre essas podem ser citadas:

- A utilização de aplicações legadas em diferentes domínios, para atestar a generalidade do processo;
- O uso de ferramentas generativas para a criação dos membros de uma Família de Produtos de Software;
- A implementação de apoio à gerência de configuração para facilitar a manutenção de Famílias de Produtos de Software;
- Técnicas alternativas de extração de características a partir de códigos legados devem ser pesquisadas para a mineração de componentes;
- O uso de técnicas de testes de equivalência para Famílias de Produtos de Software;
- O uso do processo para sistemas embutidos legados implementados em linguagens orientadas a objetos e a possível utilização de tecnologias atuais para a confecção de componentes;
- O uso de padrões de projeto e arquiteturais na implementação dos componentes e do *gateway;*

Referências Bibliográficas

ABRAHAMSSON P. **Commitment Nets in Software Process Improvement.** Annals of Software Engineering, 2002, v. 14, n. 1-4, p. 407-438, Springer Netherlands.

ATKINSON, C.; BAYER, J.; BUNSE, C.; KAMSTIES, E.; LAITENBERGER, O.; LAQUA, R.; MUTHIG, D.; PAECH, B.; WUST, J.; ZETTEL, J. **Component-Based Product Line Engineering with UML.** 1st Edition, Addison-Wesley Professional, USA, 2001.

BARROCA, L.; GIMENES, I.M.S.; HUZITA, E.H.M. Conceitos básicos. In: **Desenvolvimento Baseado em Componentes: Conceitos e Técnicas**. Editora Ciência Moderna, 2005.

BASILI, V. R. **Viewing Maintenance as Reuse-Oriented Software Development**, IEEE Software, 1990, v. 7, n. 1, p. 19-25.

BECK K. **Extreme Programming Explained: Embrace Change.** 2nd Edition, Addison-Wesley Professional, USA, 2004.

BERGEY, J.; O'BRIEN, L.; SMITH, D. **Mining Existing Assets for Software Product Lines.** CMU/SEI-2000-TN-008, Software Engineering Institute, USA, 2000.

BIANCHI, A.; CAIVANO, D.; MARENGO, V.; VISAGGIO, G. **Iterative Reengineering of Legacy Systems.** IEEE Transactions on Software Engineering, 2003, v. 29, n. 3, p. 225-241.

BOSCH, J. **Design and Use of Software Architectures: Adopting and evolving a product-line approach.** 1st Edition, Addison-Wesley Professional, USA, 2000.

BOSCH, J.; RAN, A. Evolution of Software Product Families. In: **Software Architectures for Product Families.** Lecture Notes in Computer Science, 2000, v. 1951/2000, p. 168-183, Springer Berlin / Heidelberg.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **The Unified Modeling Language Reference Manual**, Addison-Wesley, USA, 1999.

CANFORA G., CIMITILE A. **Software Maintenance**, 2000. Disponível em: <http://citeseer.ist.psu.edu/495540.html>. Acesso em: 09 mai 2007

CHIKOFSKY, E. J.; CROSS, J. H. **Reverse Engineering and Design Recovery: a Taxonomy.** IEEE Software, 1990, v. 7, p. 13-17.

CLEMENTS, P.; NORTHROP, L. **Software Product Lines: Practices and Patterns.** 1st Edition, Addison-Wesley Professional, USA, 2001

CRNKOVIC I. **Component-based software engineering for embedded systems.** Proceedings of the 27th International Conference on Software engineering (ICSE'2005). ACM SIGSOFT, 2005, p. 712-713. ACM Press, New York, USA.

CZARNECKI, K.; EISENECKER, U.W. **Generative Programming: Methods, Tools, and Applications.** Addison-Wesley Professional, USA, 2000.

FAYAD, M.; SCHMIDT, D.; JOHNSON, R. **Building Application Frameworks: Object-Oriented Foundations of Framework Design.** 1st Edition, John Wiley & Sons, USA, 1999.

FOWLER, M.; BECK, K.; BRANT, J.; OPDYKE, W. **Refactoring: Improving the Design of Existing Code.** 1st Edition, Addison-Wesley Professional, USA, 1999.

GOTTERBARN D. **UML and Agile Methods: In Support of Irresponsible Development.** ACM SIGCSE Bulletin, Column Thinking Professionally, 2004, v. 36, n. 2, p. 11-13, June.

GRAAF, B.; LORMANS, M.; TOETENEL H. **Embedded Software Engineering: The State of the Practice.** IEEE Software, 2003, v. 20, p. 61-69.

GRAHAM, I. (2000). **Object-Oriented Methods**: Principles and Practice. 3rd Edition, Addison-Wesley Professional, USA

HEINEMAN, G. T.; COUNCILL, W. T. **Component Based Software Engineering: Putting the Pieces Together.** 1st Edition, Addison-Wesley Professional, USA, 2001.

IEEE Std. **Standard for Software Maintenance.** IEEE Std 1219-1998, IEEE Computer Society Press, Los Alamitos, CA, 1998.

JACOBSON, I.; GRISS, M.; JONSSON, P. Software Reuse: Architecture, Process and Organization for Business Success. 1st edition, Addison-Wesley Professional, USA, 1997.

KANG, K.; COHEN, S.; HESS, J.; NOVAK, W.; PETERSON S. Feature-Oriented Domain Analysis (FODA): Feasibility Study. CMU/SEI-90-TR-21, 1990, Software Engineering Institute, USA.

KANG, K.; KIM, S.; LEE, J.; KIM, K.; SHIN, E.; HUH, M. **FORM: A feature-oriented reuse method with domain-specific reference architectures.** Annals of Software Engineering. Lecture Notes in Computer Science, 1998, v. 5, n. 0, p. 143-168, Springer Netherlands.

KANG K.; KIM M.; LEE J.; KIM B. **Feature-Oriented Re-engineering of Legacy Systems into Product Line Assets: a Case Study.** Lecture Notes in Computer Science, 2005, v. 3714/2005, p. 45 - 56, Springer Berlin / Heidelberg.

KRUEGER, C. W. **Easing the Transition to Software Mass Customization**. Lecture Notes in Computer Science, 2002, v. 2290/2002, p. 282, Springer Berlin / Heidelberg.

LEVESON N. G.; WEISS K. A. **Making embedded software reuse practical and safe.** ACM SIGSOFT Software Engineering Notes. Nov, 2004, v. 29, n. 6, p. 171-178. ACM Press, New York, USA.

MATINLASSI, M. **Comparison of Software Product Line Architecture Design Methods: COPA, FAST, FORM, KobrA and QADA.** Proceedings of the 26th International Conference on Software Engineering (ICSE'04), 2004, p. 127-136.

MEHTA A., HEINEMAN G.T. **Evolving legacy system features into fine-grained components.** Proceedings of the 24th International Conference on Software Engineering, Orlando, FL. Technical papers: software maintenance, 2002, p. 417-427. ACM Press, New York, USA.

MYLLYMÄKI, T.; KOSKIMIES, K.; MIKKONEN, T. **Structuring Product-Lines: A Layered Architectural Style.** Lecture Notes in Computer Science, 2002, v. 2425/2002, p. 482, Springer Berlin / Heidelberg.

NIEMELÄ E.; IHME T. **Product Line Software Engineering of Embedded Systems.** Proceedings of the 2001 Symposium on Software Reusability (SSR'01), 2001, p. 118-125, Ontário, Canadá. ACM Press, New York, USA.

O'BRIEN, L. **Reengineering**. Apresentação patrocinada pelo departamento de defesa dos EUA. Carnegie Mellon University Pittsburgh, USA, 2005. Disponível em <http://www.cs.cmu.edu/~aldrich/courses/654-sp05/handouts/MSE-Reeng-05.pdf>. Acesso em: 09 set. 2006.

OLSEM M.R. **An incremental approach to software systems re-engineering.** Journal of Software Maintenance: Research and Practice, 1998, v. 10, n. 3 , p. 181 – 202.

PAIGE R.F., WANG X., STEPHENSON Z.R., BROOKE P.J. **Towards an Agile Process for Building Software Product Lines.** XP2006, LNCS 4044, 2006, p. 198-199, Springer-Verlag Berlin Heidelberg.

PALMER S. R., FELSING J. M. **A Practical Guide to Feature-Driven Development**, The Coad Series, 1st Edition, Prentice Hall PTR, USA, 2002.

PARNAS, D. L. **Software Aging.** Proceedings of the 16th International Conference on Software Engineering, IEEE Computer Society Press, Los Alamitos, CA, 1994, pp. 279-287.

PIGOSKI, T. M. **Practical Software Maintenance - Best Practices for Managing Your Software Investment**. John Wiley & Sons, New York, NY, 1997.

PRESSMAN, R. S. **Engenharia de Software**, Editora McGraw-Hill, USA, 2002.

PRESSMAN, R. S. **Engenharia de Software**, Editora McGraw-Hill, USA, 2006.

RADINGER W., GOESCHKA K.M. **Agile Software Development for Component Based Software Engineering.** ACM OOPSLA'03, 2003, p. 300-301.

RAMOS, M. A.; PENTEADO, R. A. D. **Princípios Ágeis Aplicados à Reengenharia de Software.** In: I Workshop de Desenvolvimento Rápido de Aplicações (WDRA'2007) - VI Simpósio Brasileiro de Qualidade de Software (SBQS'2007), 2007, Porto de Galinhas - PE. Disponível em: <http://reuse.cos.ufrj.br/wdra2007/images/artigos/30398.pdf>.

RAMOS, M. A.; PENTEADO, R. A. D. **Uma Abordagem de Reengenharia Incremental para Manutenção de Software Embutido.** In: XXXIII Conferência Latino-americana de Informática (CLEI'2007), 2007, San Jose, Costa Rica.

RIEBISCH, M.; BÖLLERT, K.; STREITFERDT, D.; PHILIPPOW, I. **Extending Feature Diagrams with UML Multiplicities**. 6th World Conference on Integrated Design & Process Technology (IDPT2002), USA, 2002.. Disponível em: < http://www.theoinf.tu-ilmenau.de/~riebisch/publ/IDPT2002-paper.pdf>. Acesso em: 29 ago. 2006.

SEACORD, R.C.; PLAKOSH, D.; LEWIS, G. A. **Modernizing Legacy Systems: Software Technologies, Engineering Processes, and Business Practices.** 1st Edition, SEI-Series in Software Engineering, Addison-Wesley Professional, USA, 2003.

SEI Software Engineering Institute. <http://www.sei.cmu.edu>.

SHAW, M.; GARLAN, D. **Software Architecture: Perspectives on an Emerging Discipline**. Prentice Hall, USA, 1996.

SOCHOS, P.; PHILIPPOW, I.; RIEBISCH, M. **Feature-Oriented Development of Software Product Lines: Mapping Feature Models to the Architecture.** Lecture Notes in Computer Science, 2004, v. 3263/2004, p. 138-152.

SOCHOS, P.; RIEBISCH, M.; PHILIPPOW I. **The Feature-Architecture Mapping (FArM) Method for Feature-Oriented Development of Software Product Lines**. 13th Annual IEEE International Symposium and Workshop on Engineering of Computer Based Systems (ECBS'2006), 2006, p. 308 - 318.

STOJANOVIC Z., DAHANAYAKE A., SOL H. **Component-Oriented Agile Software Development.** XP2003. LNCS 2675, 2003, p. 315-318, Springer-Verlag Berlin Heidelberg.

SZTIPANOVITS J. **Software for Embedded Systems: Opportunities and Challenges.** Proceedings of the Sixth IEEE Real Time Technology and Applications Symposium (RTAS' 2000), 2000, p.2, IEEE Computer Society Washington, DC, USA

TENNENHOUSE, D. **Proactive computing**. Communications of the ACM, 2000, v. 43, n. 5, p. 43 - 50, ACM Press, New York, USA.

TILLEY, S.; SMITH, D. **Coming Attractions in Program Understanding**. CMU/SEI-96-TR-019, 1996, Software Engineering Institute, USA.

VAHID, F.; GIVARGIS, T. **Embedded System Design: A Unified Hardware/Software Introduction.** John Wiley & Sons, USA, 2002.

VERIFONE BR. **The Way to Pay**. Disponível em: <http://www.verifone.com.br>. Acesso em 29 ago. 2006.

WEIDERMAN, N.; NORTHROP, L.; SMITH, D.; TILLEY, S.; WALLNAU, K. **Implications of Distributes Object Technology for Reengineering**, CMU/SEI-97-TR-005, 1997, Software Engineering Institute, USA.

WEISS, D. M.; LAI C. T. R. **Software Product-Line Engineering: A Family-Based Software Development Process.** Addison-Wesley Professional, USA, 1999.

ZIADI, T.; JÉZÉQUEL, J-M.; FONDEMENT, F. **Product line derivation with UML**. In Proceedings Software Variability Management Workshop, University of Groningen, Department of Mathematics and Computing Science, 2003. Disponível em: <http://www.irisa.fr/triskell/publis/2003/Ziadi03b.pdf>. Acesso em 28 ago. 2006.