Instituto de Pesquisas Tecnológicas do Estado de São Paulo

ANA LÚCIA MIZOBE SATO

# Roteiro para a criação de Ambiente baseado em componentes para família de produtos

São Paulo

2007

ANA LÚCIA MIZOBE SATO

# Roteiro para a criação de Ambiente baseado em componentes para família de produtos

Dissertação apresentada ao Instituto de Pesquisas Tecnológicas do Estado de São Paulo - IPT, para obtenção do título de Mestre em Engenharia da Computação.

Área de concentração: Engenharia de *Software*

Orientador: Prof. Dr. Reginaldo Arakaki

São Paulo

Agosto/2007

Ficha Catalográfica
Elaborada pelo Departamento de Acervo e Informação Tecnológica – DAIT
do Instituto de Pesquisas Tecnológicas do Estado de São Paulo - IPT

S253r Sato, Ana Lúcia Mizobe
Roteiro para a criação de ambiente baseado em componentes para família de produtos. / Ana Lúcia Mizobe Sato. São Paulo, 2007.
93p.

Dissertação (Mestrado em Engenharia de Computação) - Instituto de Pesquisas Tecnológicas do Estado de São Paulo. Área de concentração: Engenharia de Software.

Orientador: Prof. Dr. Reginaldo Arakaki

1. Reuso 2. Engenharia de software 3. Produto de software 4. Qualidade de software 5. Tese I. Instituto de Pesquisas Tecnológicas do Estado de São Paulo. Coordenadoria de Ensino Tecnológico II. Título

07-181 CDU 004.415(043)

# Agradecimentos

Aos meus pais, pelo amor e orientação que sempre me deram.

Ao meu marido Sidnei, pelo apoio e incentivo nos momentos necessários e ao meu querido filho André que foi concebido durante o desenvolvimento desse trabalho.

Aos meus gerentes, Maria do Carmo e Geraldo pelo incentivo e apoio concedidos.

Aos meus colegas, Marco, Mercides e Sayuri pelo auxílio e troca de idéias no decorrer do desenvolvimento desse trabalho.

Agradeço especialmente ao orientador Prof. Reginaldo, pelos conhecimentos transmitidos e pela paciência e incentivo nas horas necessárias.

E por fim, agradeço aos funcionários do IPT, que sempre se mostraram prestativos e aos colegas do mestrado pelo incentivo, troca de experiências e ajuda mútua nos tantos trabalhos que foram desenvolvidos durante o curso.

# Resumo

O mundo dos negócios se apresenta cada vez mais competitivo, complexo e dinâmico, o que torna as empresas cada vez mais dependentes de tecnologia e de sistemas de *software* para operacionalizar seus processos e viabilizar sua gestão. No entanto, manter e evoluir esses sistemas muitas vezes se torna caro e difícil, nem sempre atendendo às expectativas de prazo e qualidade pretendidas por essas empresas.

Como conseqüência desse cenário, a indústria de *software* tem evoluído visando ao desenvolvimento e á manutenção dos sistemas com maior qualidade e menor prazo e custo. Nesse sentido, o reuso tem sido uma das principais técnicas utilizadas para alcançar tais objetivos e também tem evoluído, expandindo o conceito de reuso de códigos e executáveis, para o de reuso em um nível mais amplo como de conhecimento, de documentações e de arquitetura de *software* entre outros. O reuso pode e convém ser utilizado em todo o processo de desenvolvimento de *software*.

Esse trabalho visa à definição de um roteiro para a criação de um Ambiente para o desenvolvimento de uma família de produtos que utiliza o reuso em todas as suas fases, em maior ou menor grau. O roteiro consolida o reuso de conhecimento, de documentação, de arquitetura de *software* e de códigos, visto que, uma vez criado o Ambiente, o mesmo é complementado com as especializações dos diversos produtos que venham a pertencer a essa família, o que leva à derivação de várias aplicações desse Ambiente.

**Palavras-Chave:** Reuso; Roteiro de desenvolvimento de *software*; Família de Produto; Ambiente de *Software.*

# Abstract

The business world has been getting more and more competitive, complex and dynamic every day, making the companies more dependent on technology and on software systems to operate their processes and to manage their business. However the maintenance and the evolution of these systems are hard and expensive, and sometimes the time and quality expectations are not attended.

Consequently, the software industry has been evolving in order to aim the development and maintenance of the systems with a higher quality level and less time and costs**.** For this reason, the reuse has been one of the main techniques used to get these goals and has been evolving as well, expanding the concept of code and component reuse, for a wider level of reuse such as knowledge, documentation and software architecture. The reuse should and must be used in whole process of software development.

The goal of this study is to define a route in order to create an Environment to develop a product family using the reuse in all phases, on a large or a small degree. The process consolidates the reuse of knowledge, documentation, software architecture and components, because once the Environment is created, it is complemented with the specializations of the products of this family, generating lots of application systems from this Environment.



**Key words:** Reuse; Software Development Process; Product Family; Environment Software.

# Lista de Ilustrações

# Lista de Tabelas

# Lista de Abreviaturas

| | |
|---|---|
| CDC | Crédito Direto ao Consumidor |
| COTS | *Commercial off-the-shelf* |
| DBC | Desenvolvimento Baseado em Componentes |
| IEC | *the International Electrotechnical Commission* |
| IEEE | *Institute of Electrical and Electronics Engineers* |
| ISO | *the International Organization for Standardization* |
| UML | *Unified Modeling Language* |

Sumário

# Capítulo 1 – Introdução

A influência que a tecnologia tem, atualmente, no dia-a-dia das pessoas e principalmente no mundo dos negócios das mais diversas áreas é muito grande. Em decorrência desse fato, a indústria de *software* é uma área que tem sofrido uma demanda crescente de desenvolvimento de *softwares* cada vez mais abrangentes e mais sofisticados, o que exige uma evolução em suas técnicas e controles gerenciais utilizadas no processo de desenvolvimento de *software*.

Segundo PRESSMAN (2002), o *software* de computadores tornou-se uma força propulsora, visto que é um dos motores que dirigem a tomada de decisões nos negócios, é um fator chave que diferencia os produtos e serviços modernos e está embutido em sistemas de todas as naturezas.

A crescente importância de *software*, principalmente no contexto do mundo dos negócios, vem acompanhada de um crescente aumento de sua complexidade e também da dinâmica de seu ciclo de vida. Esse aumento de complexidade e de dinâmica é resultado de demandas geradas por necessidades do mercado, por necessidades de acompanhamento e controle dos negócios envolvidos e também pela própria dinâmica e evolução tecnológica existente.

A dinâmica do mercado atual exige agilidade por parte das empresas para atender às novas necessidades e se adaptar às novas situações. O lançamento de novos produtos, na maioria das vezes, ocorre em um curto espaço de tempo para não se perder a oportunidade do ineditismo do produto ou para acompanhar os concorrentes e não perder espaço e posições no mercado. Em muitas áreas de atuação, a empresa necessita oferecer uma carteira de produtos e serviços bastante diversificada para atender aos diferentes grupos de clientes.

Existem ainda necessidades de mudanças e adaptações às exigências dos órgãos reguladores e fiscais, do governo e de auditoria interna para acompanhar, controlar e fiscalizar o cumprimento das exigências legais e de qualidade dos processos do negócio em questão.

O dinamismo e a variedade de tecnologias que surgem a cada dia que, por um lado, proporcionam condições de serem dadas soluções cada vez mais rápidas e inovadoras, por outro lado, geram necessidades de adaptações de sistemas já existentes a essas novas tecnologias.

Para atender às necessidades com todo o dinamismo relatado anteriormente, o *software* precisa estar cada vez mais preparado para se adaptar rapidamente e suportar as mudanças impostas pelas necessidades dos negócios e das mudanças tecnológicas. Para isso, o *software* precisa ser concebido com uma estrutura sólida e de qualidade, e o passo inicial para a fase de contrução de *software* que proporciona solidez e qualidade é a definição de uma arquitetura que vai direcionar a estrutura da solução. Segundo GARLAN (2000), uma boa arquitetura ajuda a garantir que um sistema irá satisfazer seus requisitos chaves de desempenho, confiabilidade, portabilidade, escalabilidade e inter-operabilidade. Por isso, tem recebido uma maior atenção como um campo da engenharia de *software*.

## 1.1 – Motivação

Para atender às necessidades de seus requisitantes, que surgem com uma rapidez e urgência cada vez maiores, o *software* precisa funcionar e trabalhar adequadamente, ser desenvolvido por equipes, de forma eficiente e rápida e ser flexível a mudanças. Para isso é preciso buscar a melhoria continua no processo de desenvolvimento de *softwares* e, conseqüentemente, no seu produto final.

Outro aspecto que deve ser considerado é o custo relacionado ao desenvolvimento, à manutenção e à operação de *software,* uma vez que os valores envolvidos em tecnologia são representativos nos custos e investimentos totais das empresas.

Em vista desses fatos, o conceito de qualidade de *software* tem evoluído bastante nessas últimas décadas, saindo do patamar de qualidade dos requisitos externos e visíveis que até então eram considerados. A qualidade de um *software* tem sido avaliada principalmente pelos seus requisitos internos, pois são eles que possibilitam uma solução e principalmente evolução com agilidade, flexibilidade e eficiência nos resultados finais que são esperados por seus usuários e consumidores. Esses aspectos afetam diretamente os prazos e os custos de desenvolvimento de um *software*, e esses dois fatores são muito importantes na avaliação e decisões em um processo de desenvolvimento de *software*.

O reuso é um importante fator para atender às necessidades de qualidade e prazo no desenvolvimento de *software*. Segundo ESTUBLIER e VEGA (2005), o reuso tem sido um

grande objetivo em engenharia de *software,* desde que promete grandes ganhos em produtividade, qualidade e prazo no desenvolvimento de *software*.

O conceito de reuso tem se expandido, saindo do conceito de reuso de código e componentes e passando para níveis mais amplos como reuso de conhecimento, de documentações e de arquitetura entre outros, ampliando ainda mais seus benefícios.

O cenário apresentado na introdução desse capítulo, segundo o qual a importância e dependência do *software* para o dia-a-dia das pessoas e principalmente do mundo corporativo é cada vez maior, bem como a necessidade de desenvolver *softwares* com maior rapidez e qualidade e menor prazo e custo levaram a motivação do desenvolvimento desse trabalho, focado no processo de desenvolvimento de *software* com reuso dos mais variados tipos.

Na área bancária predominam sistemas na plataforma *mainframe,* onde soluções de software com reuso é pouco utilizada. Essa baixa utilização é ocasionada por alguns fatores:

- *Fatores culturais:* os desenvolvedores do *mainframe* trazem costumes anteriores á divulgação das técnicas de reuso;
- *Fatores organizacionais*: as áreas de desenvolvimento de *software* não estão estruturadas de forma a estimular e facilitar o reuso nas soluções;
- *Fatores técnicos*: as tecnologias de soluções que induzem e facilitam o reuso são mais voltadas para a baixa plataforma.

## 1.2 – Objetivo

O objetivo desse trabalho é propor um roteiro para a criação de uma solução sistêmica que permita a criação de um Ambiente para desenvolvimento de *software* baseado em componentes para uma determinada família de produtos. Pretende-se mostrar as vantagens que uma solução nessa linha pode trazer, dentre as quais se destaca o reuso dos mais variados níveis em cada fase do roteiro. E um aspecto importante, é que o roteiro pode ser aplicado em soluções *mainframe*. Espera-se, portanto, que seja um fator incentivador para o aumento do reuso em soluções dessa plataforma.

O destaque ao reuso se justifica pelos benefícios que ele pode trazer no que se refere às características de qualidade e custo no processo de desenvolvimento de *software.*

Em soluções para família de produtos, o reuso ocorre de forma intensa e em vários níveis, como será apresentado no Capítulo 3 em cada fase do roteiro, o que constitui a justificativa para a criação de um roteiro visando a atender uma família de produtos.

O roteiro proposto é exemplificado em uma aplicação prática para a solução de um Ambiente para uma família de produtos de crédito bancário.

### 1.3 – Resultados Esperados e Contribuições

Espera-se que o roteiro apresentado seja simples no entendimento, bem como na aplicação, não diferindo muito das atividades já conhecidas do modelo Clássico de desenvolvimento de *software* no qual existem as fases de levantamento de necessidades, de modelagem da solução, de desenvolvimento da solução, de testes e de implantação.

O diferencial do roteiro apresentado está na inclusão de atividades e de artefatos necessários para incorporar a aplicação do reuso em cada etapa do roteiro e com o objetivo de atender a uma família de produto.

Espera-se que o roteiro possa contribuir para um processo de desenvolvimento de *software* onde o reuso seja um objetivo e que a aplicação desse roteiro, na sua forma original ou com as adaptações necessárias, gere sistemas que atendam seus requisitos com qualidade e com menor custo.

### 1.4 – Metodologia de Pesquisa

A metodologia de pesquisa utilizada para o desenvolvimento desse trabalho envolveu as seguintes atividades:

- Definição inicial do tema do trabalho a ser desenvolvido juntamente com o orientador;
- Definição dos principais assuntos envolvidos a serem explorados;
- Pesquisa do que já foi produzido sobre os assuntos de interesse. O ato de pequisar ocorreu durante várias fases do trabalho e muitas vezes um material de pesquisa apontava para outro, enriquecendo o processo;

- Inclusão de outros assuntos que se tornaram necessários para dar completude e embasamento teórico á proposta inicial;
- Revisão do tema inicialmente escolhido, para retratar melhor o objetivo do trabalho e seu escopo;
- Escrita do trabalho com reuniões periódicas com o orientador.

Uma observação importante é que no processo de pesquisa, devido á imensidão de informações disponíveis, é preciso usar critérios de seleção, pois caso contrário, a vasta quantidade de informações pode levar a um desvio dos principais assuntos, perdendo o foco no escopo e tema do trabalho.

## 1.5 – Organização do Trabalho

Os próximos capítulos desse trabalho estão divididos da seguinte forma:

Capítulo 2 – Fundamentação Teórica

Nesse capítulo, são apresentados os principais conceitos abordados ao longo do trabalho os quais servem para dar subsídios e entendimento à solução proposta. Os principais conceitos se referem à qualidade de *software*, à arquitetura de *software*, ao reuso e ao Ambiente, bem como são apresentadas algumas metodologias e técnicas recentes que estão sendo desenvolvidas com o objetivo de permitir o desenvolvimento de *softwares* com mais qualidade, menor custo e menor tempo de desenvolvimento, como o Desenvolvimento Baseado em Componentes e Família de Produtos.

Capítulo 3 – Roteiro para a criação de um Ambiente baseado em componentes para família de produtos.

Nesse capítulo, é apresentado e detalhado o roteiro proposto para a criação de um Ambiente baseado em componentes para família de produtos.

Capítulo 4 – Exemplo de aplicação do roteiro

Nesse capítulo, é apresentado um exemplo de aplicação do roteiro proposto no capítulo 3, visando à criação de um Ambiente para uma família de produto de crédito bancário e à implantação de duas aplicações para atender a diferentes produtos.

Capítulo 5 – Considerações Finais

Nesse capítulo, são apresentadas as considerações finais sobre o trabalho, enfatizando a aplicabilidade e os benefícios da proposta do roteiro apresentado.

# Capítulo 2 – Fundamentação Teórica

Nesse capítulo, apresentam-se os conceitos necessários para o entendimento e a evolução do trabalho, com a devida fundamentação teórica e base de conhecimento para a proposta do roteiro a ser apresentado.

## 2.1 – Conceitos sobre qualidade de *software*

É cada vez maior a dependência que as empresas têm de tecnologia, mais especificamente de aplicações de *software,* para tomadas de decisões, operação, controle e gestão de seus negócios. Assim como é fato a crescente complexidade dos *software*s solicitados e do peso que a tecnologia tem nos custos e nos investimentos das empresas. Esse cenário torna clara a necessidade, bem como a exigência da qualidade do produto final de tecnologia a ser entregue.

Segundo PRESSMAN (2002), definir o conceito de qualidade de *software* não é uma tarefa fácil, visto que *software* não apresenta características e atributos mensuráveis, por tratar-se de uma entidade intelectual. Mesmo assim, existem várias definições, e algumas delas são apresentadas a seguir.

### 2.1.1 – Definições de qualidade de *software*

Qualidade de *software* é um tema bastante abrangente e sua avaliação depende das expectativas dos envolvidos com ele. Para um **usuário final** da aplicação a ser desenvolvida, a qualidade é obtida com um produto final que atenda suas necessidades com fácil utilização e com poucas incidências de falhas. Para o **desenvolvedor**, qualidade se refere à facilidade de se prover manutenções nos sistemas, à facilidade de testes, à flexibilidade na implementação de novas funcionalidades e, para o **gerente do projeto,** a aplicação tem que ser eficiente em termos de utilização de recursos e de ser desenvolvida em um prazo menor e com menor custo.

2.1.2 – ISO/IEC 9126-1

Para avaliação da qualidade de *software*, assim como de outras áreas, é necessária a definição de alguns padrões internacionais de requisitos, modelos e métricas. Para a qualidade de *software,* um dos padrões existentes é o ISO/IEC 9126-1- *Software Engineering – Product Quality*. A ISO/IEC 9126-1 (2001) apresenta quatro partes: Modelo de Qualidade, Métricas Externas, Métricas Internas e Métricas de Qualidade em Uso.

Para o contexto desse trabalho será explorada a parte 1 – Modelos de Qualidade, na qual são definidos dois Modelos de Qualidade de produto de *software:*

- *Modelo de Qualidade Interna e Externa* – é a totalidade de características do produto de *software* do ponto de vista interno e externo. Nesse modelo, a qualidade é medida e avaliada com relação aos requisitos de qualidade internos e externos quando o *software* é executado, enquanto ele está sendo testado em um ambiente, com dados simulados e usando métricas externas;
- *Modelo de qualidade em uso* – é a visão de qualidade do produto de *software* do ponto de vista do usuário, quando esse produto é usado em um ambiente e em um contexto de uso específico. Ela mede o quanto os usuários podem atingir seus objetivos em um determinado ambiente e não as propriedades do *software* .

Para o modelo de qualidade interna e externa, são definidas seis características a serem consideradas:

- *Funcionalidade* – refere-se à capacidade de prover funções que atendam às necessidades explícitas e implícitas, quando o s*oftware* estiver sendo utilizado sob condições específicas;
- *Confiabilidade* – refere-se a atributos que garantam a execução da aplicação com nível de desempenho especificado, quando usada em condições específicas;
- *Usabilidade* – refere-se à capacidade de a aplicação ser entendida, aprendida e utilizada pelo usuário final de forma amigável e simples, quando usada em condições específicas;

- *Eficiência* – refere-se às características de o *software* ser executado com desempenho apropriado e compatível com os recursos utilizados, sob condições específicas;
- *Facilidade de Manutenção* – refere-se à capacidade de o *software* sofrer manutenções de reparo de falhas e melhorias ou adaptações devido a mudanças no ambiente e nos seus requisitos ou especificações funcionais;
- *Portabilidade* – refere-se à facilidade pela qual o *software* pode ser transferido de um sistema computacional ou de um ambiente para outro.

Para o modelo de qualidade em uso, são definidas quatro características a serem consideradas:

- *Eficácia* – refere-se à capacidade de o usuário alcançar objetivos específicos com exatidão e de forma completa em um contexto de uso específico;
- *Produtividade* – refere-se à capacidade de permitir ao usuário utilizar uma quantidade apropriada de recursos em um contexto de uso específico em relação à eficácia obtida;
- *Segurança* – refere-se à capacidade do produto de *software* de apresentar níveis aceitáveis de risco de danos às pessoas, aos negócios, aos *softwares,* às propriedades e ao ambiente em um contexto de uso específico;
- *Satisfação* – refere-se à capacidade de satisfazer o usuário com os resultados obtidos, na sua interação com o produto e também nas atitudes envolvidas com o uso do produto em um contexto de uso específico.

### 2.1.3 – Outros atributos de qualidade de *software*

BASS, CLEMENTS e KAZMAN (2003) consideram outros atributos de qualidade de sistemas:

- *Disponibilidade* – refere-se ao tratamento de falhas de sistemas e suas conseqüências. Deve ser considerado o modo como as falhas são detectadas, com

qual freqüência elas ocorrem, o que acontece quando elas ocorrem, por quanto tempo o sistema pode ficar fora de operação, como prevenir falhas e que tipo de comunicação deve ser feita quando elas ocorrerem;

- *Segurança* – é a capacidade de o sistema não permitir acesso de usuários não autorizados. Pode ser caracterizado com um sistema que provê as seguintes características:
  - ✓ *Não-repúdio* – uma transação não pode ser negada por uma das partes envolvidas, visto ser essa a garantia que impede que uma entidade participante de uma operação negue essa participação;
  - ✓ *Confidencialidade* – os dados e serviços estão protegidos para acessos não autorizados;
  - ✓ *Integridade* – os dados e serviços são entregues como pretendidos e seus valores só podem ser alterados por operadores autorizados;
  - ✓ *Segurança/garantia* – as partes envolvidas em uma transação são as definidas e somente elas;
  - ✓ *Disponibilidade* – o sistema fica disponível para uso;
  - ✓ *Auditagem* – o sistema apresenta informações de rastreamento suficientes para que suas atividades possam ser reconstruídas.
- *Facilidade de testes* – o sistema deve ter suas falhas demonstradas facilmente por testes. Pelo menos 40% do custo de desenvolvimento são tomados por testes.

Segundo D´SOUZA e WILLS (1998), uma empresa precisa permanecer competitiva, com seus negócios mudando continuamente, provendo novos produtos e serviços e, para isso, seu processo operacional deve ser adaptado conforme essas novas necessidades. A grande maioria desses processos operacionais é suportada por um ou mais sistemas de *software,* sendo que eles devem ser flexíveis para não somente serem alterados rapidamente, mas também para proverem várias possibilidades de soluções. Portanto, flexibilidade também é um importante aspecto de qualidade a ser considerado.

Os aspectos de qualidade referentes à *software* são muito variados e muitas vezes conflitantes entre si. O ideal é que, no desenvolvimento de um projeto de *software,* todos os requisitos de qualidade sejam considerados, porém, para cada projeto, devem existir requisitos de qualidade que precisam ser priorizados na solução a ser dada em função das necessidades específicas ou características do projeto. Esses requisitos necessitam estar bem claros para os envolvidos desde o início do projeto, pois vão influenciar as decisões e a solução a ser dada. Cada fase do processo de desenvolvimento de *software* apresenta características de qualidades inerentes a ela, portanto cada fase do roteiro apresentado também as tem.

### 2.1.4 – Contribuições para o trabalho

Uma solução de *software* com qualidade é sempre um objetivo desejado, mas nem sempre alcançado, pelo menos em todos os aspectos. Para obter uma solução com as características de qualidade em níveis desejados, os atributos de qualidade esperados precisam estar bem definidos, para que, em cada fase do projeto, possam ser considerados. Alguns atributos de qualidade trazem mais efeitos e impactos em determinadas fases do projeto, por isso precisam estar definidos desde o início para que não sejam esquecidos ou negligenciados no momento em que forem necessários.

## 2.2 – Arquitetura de *Software*

A definição de arquitetura de *software* é muito ampla e são descritas das mais diversas formas ao longo do tempo, justamente por sua ampla abrangência. Nesse trabalho, o foco de arquitetura que se quer dar encontra-se dentro do escopo de modelagem lógica da solução, na fase de seu projeto, quando os grandes módulos que a solução vai ter estão sendo definidos. São citadas algumas dessas visões no desenvolvimento desse ítem.

### 2.2.1 – Projeto de *Software*

Segundo PRESSMAN (2002), o projeto de *software* situa-se no núcleo técnico da engenharia de *software* e é a primeira de três atividades técnicas – projeto, geração de código e teste, que são necessárias para construir e verificar *software*.

O projeto é a etapa na qual a qualidade é incorporada à engenharia de *software*, a qual fornece representações do *software* que podem ser avaliadas quanto à qualidade. Sem o projeto, arriscamo-nos a construir um sistema instável, que falha quando pequenas modificações são realizadas, falhas que podem ser difíceis de testar. Além de que a qualidade do sistema não pode ser avaliada até uma fase avançada do processo de *software*, quando, então, o prazo está se esgotando e muito dinheiro já foi gasto.

Existem alguns conceitos fundamentais de projeto de *software* que devem ser considerados para um bom projeto:

- *Abstração* – Permite que um problema complexo seja considerado em um nível de generalização, sem entrar em detalhes de baixo nível de solução. Cada passo do processo de *software* é um refinamento no nível de abstração da solução;
- *Refinamento* – O refinamento leva o projetista a elaborar o enunciado original, fornecendo mais e mais detalhes à medida que cada refinamento sucessivo ocorre. Abstração e refinamento são conceitos complementares. Ambos os conceitos permitem ao projetista criar um modelo completo de projeto conforme o próprio projeto evolui.
- *Modularidade* – O *software* é dividido em componentes nomeados separadamente e endereçáveis, freqüentemente chamados de módulos, que são integrados para satisfazer os requisitos do problema. É mais fácil resolver um problema complexo quando ele é dividido em partes menores gerenciáveis;
- *Arquitetura de software* – devido à sua importância no trabalho, esse conceito será abordado no item a seguir, com mais detalhes;
- *Ocultação da informação* – os módulos devem ser projetados de modo a serem independentes. Seus procedimento e dados devem ficar inacessíveis a outros módulos, visto que devem ficar acessíveis somente as informações necessárias para que os referidos módulos realizem uma função do *software*.

2.2.2 – Definições sobre Arquitetura de *Software*

Segundo PRESSMAN (2002), a arquitetura de *software* é a estrutura do sistema que abrange os componentes de *software*, as propriedades externamente visíveis desses componentes e as relações entre eles. A arquitetura não é *software* operacional. Ao contrário, é a representação que permite ao engenheiro de *software* analisar a efetividade do projeto em satisfazer seus requisitos declarados, considerar alternativas arquiteturais num estágio em que fazer modificações no projeto ainda é relativamente fácil e reduzir os riscos associados à construção do *software*.

Segundo GARLAN (2000), a arquitetura de *software* tipicamente faz o papel de ponte entre os requisitos e a implementação, conforme a figura 2.

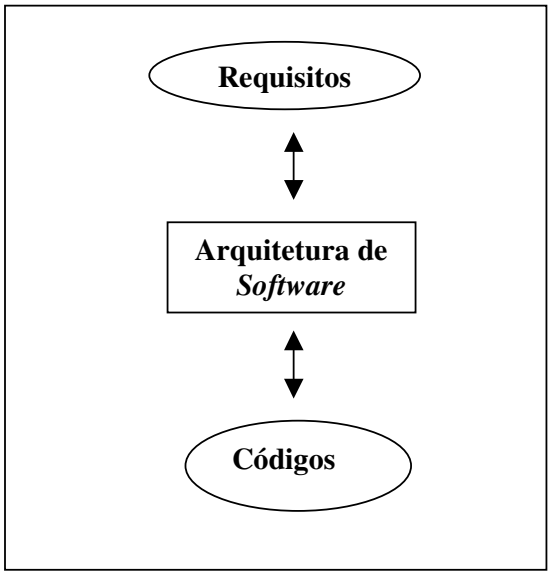



Figura 1 – Arquitetura de s*oftware* como ponte (GARLAN, 2000).

Para elaboração da arquitetura de *software*, pelo menos seis aspectos de seu desenvolvimento devem ser considerados:

- *Entendimento* - Simplifica nossa habilidade para compreender um grande sistema, apresentando-o em um nível de abstração tal que permita um fácil entendimento;
- *Reuso* **-** Descrições da arquitetura permitem o reuso em múltiplos níveis. De bibliotecas de componentes, de *frameworks* nos quais os componentes podem ser integrados, de padrões de modelagem e existindo trabalhos mais recentemente de domínios específicos;
- *Construção* - A arquitetura fornece um rascunho parcial para desenvolvimento ao indicar os principais componentes e as dependências entre eles;
- *Evolução* - Arquitetura de *software* define as dimensões do sistema, tornando explícito seu escopo, o que permite aos desenvolvedores entender a extensão das mudanças que venham a ocorrer, bem como propicia uma estimativa de custo mais apurada;
- *Análise* - A Arquitetura fornece oportunidades para análise de consistências de checagem do sistema, conformidade com as *constrains* impostas pelo estilo da arquitetura, conformidade com os requisitos de qualidade, análise de dependência e análise de domínio específico para arquitetura construída;
- *Gerenciamento* - Experiências têm mostrado que projetos de sucesso têm sido alcançados com uma arquitetura de *software* viável. Uma avaliação crítica de uma arquitetura permite um entendimento mais claro dos requisitos, das estratégias de implementação e dos riscos potenciais.

Segundo MENDES (2002), o desenvolvimento de *software* no nível arquitetural compreende questões estruturais, entre as quais se destacam:

- Seleção de alternativas de projeto;
- Escalabilidade e desempenho;
- Organização e estrutura geral de controle;

- Protocolos de comunicação e sincronização;
- Atribuição de funcionalidade a componentes de projeto.

A idéia básica de arquitetura de *software* é a de que um sistema de *software* em elevado nível de abstração pode ser descrito por meio de subsistemas ou de componentes distintos, sendo esses inter-relacionados.

O projeto arquitetural é uma atividade interativa na quais várias decisões de projeto importantes são tomadas. Os requisitos arquiteturais juntamente com os cenários de uso são utilizados na tomada de decisões, sendo que um modelo arquitetural é utilizado para descrever um sistema decomposto em um conjunto de módulos ou de componentes conectados.

A arquitetura é resultado de um conjunto de decisões de negócios e de decisões técnicas, visto que ela é influenciada por vários aspectos, entre os quais podemos citar os seguintes:

- *Expectativas dos envolvidos no projeto* (*stakeholders*) - Muitas vezes as expectativas dos envolvidos são conflitantes, sendo necessário que o arquiteto priorize e concilie essas expectativas, de forma a atender a todos, dentro do possível e viável;
- *Área de desenvolvimento da organização/ambiente técnico* – Se a empresa tem toda a infra-estrutura, a equipe de pessoas e as ferramentas voltadas para uma solução cliente-servidor, provavelmente a arquitetura da solução é cliente-servidor, caso contrário, a solução pode ter um custo e um risco maior;
- *Conhecimento e experiência do arquiteto* **–** O arquiteto provavelmente usa uma solução com a qual ele já obteve sucesso em implementações anteriores e evita usar soluções com que não obteve resultados satisfatórios.

Os fatores citados que influenciam a arquitetura também são influenciados por ela. Em caso de um projeto implantado com sucesso, as características arquiteturais desse projeto influenciam os próximos, direcionando a montagem de equipes especializadas nessa arquitetura, o que facilita a criação de produtos similares com rapidez e qualidade, visto que

estabelece padrões de uso para os usuários desses sistemas, gerando assim o reuso da arquitetura.

Conforme o escopo e a criticidade do negócio a ser atendido pelo *software* a ser desenvolvido, devem estar bem definidas as prioridades das características de qualidade que esse sistema deve ter, pois essa definição direciona o modelo da sua arquitetura. E as decisões tomadas no projeto da arquitetura do *software* influenciam as características de qualidade do *software* durante todo seu ciclo de vida.

Para VAROTO (2002), a importância da arquitetura pode ser resumida em três aspectos:

- *Arquitetura e a comunicação entre os participantes* - uma vez que a arquitetura representa uma abstração comum em alto nível de um sistema que serve para obter um entendimento mútuo e formar consenso entre os vários participantes do projeto, cada um deles colabora com suas visões específicas;
- *Arquitetura e a antecipação de decisões de projeto* – esse aspecto é um ponto de referência comum para as demais atividades que são executadas posteriormente a sua definição. Ele serve de guia para o projeto de implementação, testes e implantação do sistema. Para isso, precisa ter passado por extensas análises, avaliações e revisões até chegar a sua versão final, pois essa precisa estar congelada, para evitar que mudanças e decisões no meio das atividades de implementação coloquem em risco a construção do sistema;
- *Arquitetura e o reuso* - se a arquitetura estiver bem organizada e estruturada, cada componente computacional ou parte dele pode ser construído visando ao reuso. O reuso é uma característica de qualidade agregada à definição da arquitetura.

### 2.2.3 – Várias Visões de Arquitetura

Uma arquitetura é, inicialmente, uma abstração da implementação de um sistema, sendo que existem muitos modelos de arquiteturas diferentes que auxiliam no entendimento de um sistema. Cada modelo auxilia na análise de um requisito de qualidade do sistema: tanto em aspectos de execução, como desempenho, segurança, confiabilidade, como em aspectos de desenvolvimento, como facilidade de manuntenção e portabilidade.

Segundo VAROTO (2002) apresenta em seu trabalho, cada participante apresenta uma visão diferente sobre a representação da arquitetura, segundo critérios variados. Não há uma representação simples para a arquitetura: há várias representações, com diferentes perspectivas e regras. Representar a arquitetura em várias visões, utilizando vários modelos, produz um resultado mais consistente uma vez que há muita informação dissociada para retratar em um único modelo.

### 2.2.4 – Modelo "4+1" de visões de arquitetura de *software*

O desenvolvimento de um sistema complexo envolve várias pessoas e cada uma com interesses e visões distintas. O modelo de uma arquitetura de *software* visa a atender a múltiplas e concorrentes visões dos diversos envolvidos no projeto do *software*: usuários finais, desenvolvedores, engenheiros de *software*s, gerentes, entre outros. Os modelos utilizados atualmente visam a atender a todas as visões em um único modelo e acabam por não atender a nenhuma delas de forma completa. Philippe KRUCHTEN (1995) propôs o Modelo de visões "4+1" de arquitetura de *software* que engloba a visão lógica, de processo, física, de desenvolvimento e de cenários de forma distinta, como é apresentado na figura a seguir.

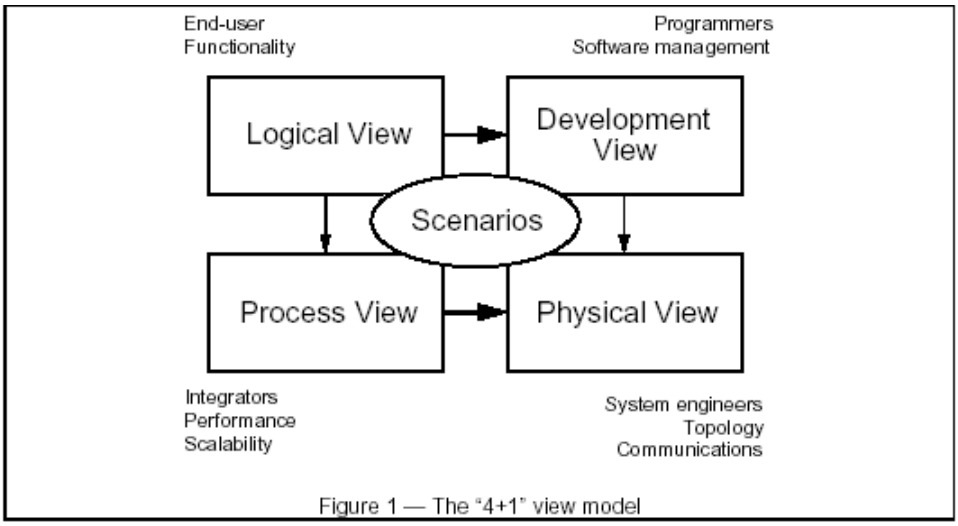



Figura 2 – "Modelo de Visões 4+1 da arquitetura de s*oftware*" (KRUCHTEN,1995).

A visão lógica deve suportar os requisitos funcionais, bem como demonstrar os serviços que o sistema deve prover para seus usuários. Essa visão tem como objetivo realizar uma análise funcional, e mais, identificar mecanismos e elementos comuns pelas várias partes do sistema. Pode ser representado por modelos de Orientação a Objetos, assim como de Análise Estruturada.

A visão de processo descreve requisitos não-funcionais como características de concorrência, sincronismo, paralelismo, tolerância á defeitos. Entende-se por processo, um conjunto de tarefas que formam uma unidade executável. O modelo que representa a visão de processo pode ser descrito em diferentes níveis de abstração, sendo cada nível direcionado para objetivos diferentes. Um processo é um grupo de tarefas que compõem uma unidade executável, que pode ser taticamente controlada e também pode ser replicada para aumentar a capacidade de processamento ou melhorar sua disponibilidade.

A visão de desenvolvimento descreve a organização estática do *software* no seu ambiente de desenvolvimento, nos módulos e nos subsistemas que compõem a solução. Essa visão é base para a alocação de recursos de *hardware* e técnicos, para alocação de recursos humanos, para previsão de custos e para o planejamento do projeto. KRUCHTEN (1995) recomenda que seja adotado um estilo em camadas para essa visão, definindo de 4 a 6 camadas de subsistemas. Cada camada com uma responsabilidade bem definida e um

subsistema de uma camada só deve se relacionar com outros da mesma camada ou da camada abaixo, visando a minimizar o desenvolvimento de interfaces complexas entre os módulos e permitir uma estratégia simples de liberações camada por camada.

A visão física descreve o mapeamento do *software* no que se refere ao *hardware* que o suporta e reflete o aspecto de sua distribuição. A referida visão deve representar os aspectos não-funcionais referentes à disponibilidade, segurança, desempenho e escalabilidade.

O Cenário coloca todas as visões juntas. Os elementos das quatro visões devem trabalhar integrados atendendo aos mais importantes cenários, que são os mais importantes requisitos. A visão do Cenário é muito similar à visão lógica, mas usa os conectores da visão de processos. Pode parecer uma redundância das demais visões, mas serve para dois propósitos importantes:

- Como um direcionamento para descobrir os elementos arquiteturais durante a definição da arquitetura;
- Como uma validação e representação depois que o modelo da arquitetura estiver completo.

Para cada visão a que se quer atender no modelo descrito anteriormente, podem ser utilizados formas e padrões diferentes e mais adequados para sua representação. Bem como podem ser utilizados padrões já existentes, sem a necessidade da criação de novos.

As várias visões não são totalmente independentes. Elementos de uma visão são conectados em elementos de outras visões, seguindo certas regras e heurísticas. Porém nem todos os sistemas precisam das “4+1” visões. Visões sem utilidades para uma determinada solução podem ser omitidas, como a visão física, se a solução tem somente um processador, e a visão de processo, se há somente um processo ou programa. Já o cenário é útil em qualquer circunstância.

2.2.5 – Contribuições para o trabalho

Uma arquitetura bem definida contribui muito para a qualidade do projeto a ser desenvolvido. É um passo inicial muito importante para alcançar uma solução final com qualidade, pois fornece a estabilidade estrutural para suportar as demais fases do projeto e, mais, para viabilizar a manutenção dessa estabilidade durante o ciclo de vida dos sistemas resultantes desse projeto.

A arquitetura da solução pode ser um grande artefato de reuso entre sistemas com características similares. E especificamente nesse trabalho, a definição de uma arquitetura visa viabilizar o reuso da parte comum da solução pelas aplicações derivadas do Ambiente criado.

Os conceitos de arquitetura definidos são utilizados na fase de Definição da Arquitetura da Solução e devem ser conhecidos e mantidos nas demais fases do projeto.

## 2.3 – Reuso

Reuso tem sido um grande objetivo para a engenharia de *software*, pois o conceito de reuso está associado ao aumento de produtividade e qualidade e à redução do tempo de entrega, gerando grande interesse na comunidade acadêmica e de negócios de *software.*

O aumento de produtividade é obtido quando um novo projeto passa a usar artefatos de *software* já prontos, sem a necessidade de desenvolvê-los desde o início. O aumento da qualidade é obtido quando esses artefatos reutilizados já foram utilizados, depurados e melhorados por outros sistemas, sem o risco de erros gerados por artefatos novos.

2.3.1 – Níveis de reuso

Reuso tem sido mais relacionado com reuso de códigos, disponibilizados por componentes ou serviços. Mas com o amadurecimento e com estudos sobre esse assunto, percebeu-se que o reuso pode ser aplicado mais amplamente do que apenas no código, pode ser aplicado reuso de conhecimento, documentos, processos, entre outros. Nesse trabalho, o reuso visa não somente ao reuso de código, mas também ao reuso de vários outros artefatos de *software*.

O requisito de reusabilidade, segundo MENDES (2002), é a característica de fazer uso de componentes já desenvolvidos, objetivando minimizar esforços em novos projetos e, conseqüentemente, custos no seu desenvolvimento. O reuso pode ter diferentes perspectivas.

Pode ser orientado a componentes, a processos ou a conhecimento (específico) de um domínio. A criação de componentes, permitindo sua reutilização em vários outros sistemas posteriores, contribui muito para o desenvolvimento rápido de sistemas, com baixo custo e com ínfimo risco de erros, à medida que os componentes são testados e utilizados por outros sistemas anteriormente.

Para LEVESON e WEISS(2004), o reuso mais problemático tem sido o de código, visto que o reuso pode ser mais efetivo e seguro quando usado em fases iniciais do processo de desenvolvimento de *software*. A codificação é uma pequena parte do processo de desenvolvimento de *software*, principalmente para *software* de controle *on-line*, sendo assim, o custo de repetir o código é insignificante comparado ao custo potencial de revalidar o código reusado. E mais, uma mudança de código segura é mais fácil quando as mudanças são feitas em fases iniciais do processo.

Segundo CRNKOVIC e LARSSON (2000), o conceito de reuso pode ser utilizado em diferentes níveis. Em um nível mais baixo, está o reuso de códigos fontes e também de pequenos e simples componentes. Em um nível intermediário, grandes componentes encapsulando funções de negócios. Finalmente, a integração de produtos completos em um sistema complexo pode ser visto em um nível mais alto de reuso. Em cada nível de reuso há demandas específicas para os componentes reusáveis, para o gerenciamento dos componentes e para o processo de integração.

ROTHENBERGER et al. (2003) definem os mais diversos artefatos de reuso: *procedures*, conhecimento, documentação, arquitetura, modelo e código. Construir um *software* reusável requer esforços adicionais comparados com um *software* não voltado para reuso, pois precisa ser construído de uma forma genérica para que possa ser utilizado nos mais variados contextos. Criar um repositório de *software* e carregá-lo com componentes reusáveis é um investimento substancial para uma organização, visto que o retorno só é obtido ao longo do tempo quando o esforço é recuperado com projetos de *software* com reuso. Portanto, a adoção de reuso de *software* envolve riscos e a escolha de uma estratégia de reuso é crucial para o seu sucesso.

Segundo Tracz (1986 apud ROSSI, 2004), um estudo relativo à reutilização de código fonte mostra os seguintes dados:

- 40% a 60% de todo código são reutilizáveis de uma aplicação para outra;

- 60% do projeto e do código sobre aplicações de negócio são reutilizáveis;
- 75% das funções de programa são comuns a mais de um programa;
- 15% dos códigos encontrados na maioria dos programas são únicos e novos para uma aplicação específica.

O reuso requer que os componentes sejam construídos de forma que sejam genéricos, isto é, não específicos para uma determinada aplicação, mas também sejam customizáveis de forma a se adaptarem a uma necessidade específica.

O reuso de componentes de *software* deve ser apoiado por um ambiente que inclua os seguintes elementos:

- Uma base de dados de componentes capaz de armazenar componentes de *software* e a informação de classificação necessária para sua recuperação;
- Um sistema de gestão de biblioteca que forneça acesso à base de dados;
- Um sistema de recuperação de componentes de *software* que permita a uma aplicação cliente recuperar componentes e serviços do servidor da biblioteca;
- Ferramentas que suportem a integração de componentes reusados numa implementação ao projeto novo.

O reuso não é um problema de ordem técnica, mas sim de processo, de organização e é cultural. Segundo D´SOUZA e WILLS (1998), em uma cultura de reuso, uma organização visa a construir e a aumentar seu capital de bens reusáveis. Como um investimento, esse capital deve ser gerenciado e cultivado. Ele requer investimento na construção desses bens, um apropriado processo de desenvolvimento e de regras, bem como treinamento e incentivos.

2.3.2 – Características importantes para reuso

Krueger (1992) apresenta quatro dimensões para caracterizar e avaliar uma tecnologia de reuso de *software*:

- *Abstração:* é uma característica essencial para qualquer técnica de reuso. É a descrição suscinta que suprime os detalhes e enfatiza as informações importantes. A abstração de *software* apresenta dois níveis: o nível mais alto se refere à especificação da abstração e o nível mais baixo, mais detalhado, é chamado de realização ou implementação da abstração;
- *Seleção:* é a dimensão que auxilia os desenvolvedores de *software* a localizar, comparar e selecionar artefatos de *software* reusáveis;
- *Especialização:* Em muitas técnicas de reuso, artefatos similares são agrupados em artefatos genéricos. Depois de selecionar para reuso um artefato genérico, o desenvolvedor especializa-o utilizando parâmetros, transformações ou alguma outra forma de refinamento;
- *Integração:* Uma tecnologia de reuso geralmente tem um *framework* de integração. O desenvolvedor usa esse *framework* para combinar uma coleção de artefatos selecionados e especializados para gerar um sistema de *software* completo.

2.3.3 – Reuso em Família de Produtos

Segundo ESTUBLIER e VEGA (2005), na prática, um reuso mais efetivo tem sido obtido em dois casos: em bibliotecas, onde componentes pequenos e genéricos podem ser encontrados; e família de produtos, onde componentes de um domínio específico podem ser construídos de diferentes formas para produzir um determinado produto. Eles acreditam que lições aprendidas pela comunidade de família de produtos podem ser aplicadas em um escopo maior. Essas lições são:

- *Obter uma descrição abstrata e estável de um problema a ser resolvido:* a maioria das soluções para família de produtos são realizadas de forma *top down*, quando a fase inicial de análise identifica as partes comums e as diferentes entre os membros da família. Essa análise permite a criação de uma arquitetura comum com um número de

pontos variáveis para permitir as diferenças. Essa arquitetura descreve o produto mais do ponto de vista de suas funcionalidades (o problema a ser resolvido) do que do ponto e vista de tecnologia (da solução). Essa característica permite a construção de uma arquitetura estável, o que a torna um dos primeiros artefatos reusáveis;

- *Identificar as variações em termos de caraterísticas da funcionalidade e não em termos de solução:* os pontos de variações estão relacionados a uma arquitetura abstrata, o que permite que ela seja compartilhada por um grande número de produtos. As definições de variações do ponto de vista das funcionalidades garantem a estabilidade da solução;
- *Reuso de alta granularidade, componentes de funcionalidade de alto nível:* A arquitetura identifica grandes conjuntos de funcionalidades comuns, os quais se tornam componentes reusáveis. Esses componentes são relativamente grandes e atendem a funcionalidades de alto nível direcionadas para a família de produto.

### 2.3.4 – Contribuições para o trabalho

O reuso é um dos principais objetivos a serem alcançados com a aplicação do roteiro apresentado. Os artefatos de reuso estão presentes em todas as fases do roteiro nos mais variados níveis, desde o mais alto e abstrato como o conhecimento, até o mais baixo, os códigos.

## 2.4 – Desenvolvimento Baseado em Componentes

O Desenvolvimento Baseado em Componentes (DBC) surgiu no final da década de 90 visando á intensificação do reuso. O DBC se preocupa com a criação de componentes que possam ser reutilizados em outras aplicações. É uma atividade com peso crescente na indústria de *software* e a principal justificativa para tal crescimento é econômica, visto ser uma abordagem que permite simultaneamente reduzir custos, diminuir prazo e aumentar a qualidade no desenvolvimento.

### 2.4.1 – Componentes

Segundo D´SOUZA e WILLS (1998), componente é um pedaço coerente de *software* que pode ser desenvolvido de forma independente e ser disponibilizado como uma unidade a ser composta, sem alterações, com outros componentes para construir algo maior.

Segundo Brown e Wallnau (1996 apud PRESSMAN, 2002), componente é uma parte não trivial de um sistema, praticamente independente e substituível, que preenche uma função clara no contexto de uma arquitetura bem definida. Exemplificando, um componente muito simples pode ser uma função matemática que processa a raiz quadrada de um número.

O componente reusável apresenta duas características fortes:

- O componente é uma entidade executável independente, seu código fonte não fica disponível e, portanto, ele não é compilado com outros componentes do sistema;
- Suas interfaces e todas as interações ocorrem por meio dessas interfaces. A interface do componente é expressa em termos de operações e seus estados internos nunca são expostos.

Está mudando a forma como sistemas de *software* de grande porte são desenvolvidos, visto que a implementação deu lugar à integração como foco. Na fundamentação dessa mudança, está a pressuposição de que há muitos pontos em comum entre vários sistemas de *software* de grande porte, o que justifica o desenvolvimento de componentes reusáveis para explorar e satisfazer esses pontos comuns.

Existem componentes comerciais prontos para uso que são denominados COTS – *commercial off-the-shelf* e cuja utilização vem crescendo no mercado de *software* onde o reuso e o desenvolvimento baseado em componentes são utilizados.

### 2.4.2 – Características do DBC

Segundo LIN e YONGSEN (2003), existem dois elementos básicos do *software* baseado em componentes: componente de *software* e modelo do componente. O componente de *software* é uma parte independente a ser entregue de funcionalidade provida de acesso para seus serviços via interfaces. Pode ser implantando de forma independente e está sujeito a ser composto por terceiras partes. Um modelo de componente é o ‘*backbone’* de um sistema

baseado em componentes. Ele fornece suporte para o desenvolvimento de componentes, composições, comunicações e implantações, entre outros artefatos.

Segundo CRNKOVIC e LARSSON (2000), um componente precisa ser genérico o suficiente para cobrir os diferentes aspectos de sua utilização. Ao mesmo tempo, ele precisa ser objetivo e simples, para servir a um requisito específico de uma forma eficiente. Desenvolver componentes para reuso requer três ou quatro vezes mais recursos que o desenvolvimento de um componente para um uso particular.

A evolução de requisitos dos produtos gera novos requisitos para os componentes. Segundo CRNKOVIC e LARSSON (2000), esse dinamismo determina o ciclo de vida do componente, sendo que, inicialmente, ganha estabilidade e, posteriormente, vai sofrendo uma degeneração e vai se tornando difícil para ser utilizado, adaptado e mantido. Quando alcança esse estágio, o componente se torna um obstáculo para um reuso eficiente e precisa ser substituído.

### 2.4.3 – Contribuições para o trabalho

Com a aplicação das técnicas do DBC, espera-se obter uma solução com facilidade de construção e manutenção e que seja flexível, uma vez que é composta por pequenos componentes com funções e interfaces específicas e mais simples. Esses conceitos são aplicados na fase de Definição do Ambiente, quando os componentes necessários são levantados e, na fase de Elaboração de Componentes, quando os componentes não encontrados são construídos.

## 2.5 – Ambiente

O termo Ambiente utilizado nesse trabalho se refere às atividades relacionadas no processo de desenvolvimento de um *Framework* para a família de produto e seus respectivos resultados, desde os requisitos de negócios, projeto lógico, projeto físico, padrões definidos até a formação da equipe de projeto, sistemas legados envolvidos e plataformas de processamento. O principal resultado do Ambiente é o *Framework* gerado, porém muitas atividades e estruturações são necessárias para criar e também manter esse *Framework*.

2.5.1 – *Framework*

Além de um conjunto de componentes, que isoladamente é apenas um código de programa, pode-se criar uma especificação e um modelo composto por componentes e armazená-los em uma biblioteca e, posteriormente, combiná-los em diferentes configurações. Assim é criado um *Framework* que é um sistema ou rotina quase pronta, precisando ser completado com rotinas específicas do contexto do sistema no qual está sendo utilizado. Um *Framework* permite atender ao requisito de reusabilidade, haja vista que ele pode ser utilizado em várias aplicações.

Um *Framework* é uma estrutura genérica que pode ser completada para criar uma aplicação ou subsistema mais específico. Segundo WILLIAMS (2001), o *Framework* não é uma aplicação completa, visto que o desenvolvedor precisa completá-lo para criar uma aplicação. Essa extensibilidade é um dos principais benefícios de um *Framework*, pois permite que sua infra-estrutura seja utilizada por múltiplas aplicações.

Segundo WILLIAMS (2001), um *Framework* tem benefícios adicionais em comparação a uma simples biblioteca de componentes. Tipicamente, quando uma aplicação usa um componente, a aplicação o chama quando necessário, ou seja, o controle do fluxo está com a aplicação. Em um *Framework*, o controle do fluxo é invertido, o *Framework* toma esse controle e decide quais partes de uma aplicação são chamadas e quando, assim como com qual método elas são chamadas. Essa característica permite que o Ambiente que contempla esse *Framework* incorpore um 'fluxo de processo' de um negócio, tornando o Ambiente uma solução completa e não parte de uma solução. Em um *Framework*, as dependências/colaborações estão embutidas, sendo assim, ele impõe um modelo de colaboração a que a aplicação tem que respeitar e se adaptar.

Um *Framework* captura as funcionalidades comuns a várias aplicações, por isso, elas devem ter algo razoavelmente grande em comum, devem pertencer a um mesmo domínio de problema.

Segundo D´SOUZA e WILLS (1998), um princípio básico que deve ser usado na definição do *Framework* é o de que as chamadas das funções partem do *Framework,* é ele que implementa a arquitetura e impõe as regras e as políticas para as aplicações.

### 2.5.2 – Outros aspectos de Ambientes

Ambientes podem representar um valor significativo para o negócio de uma empresa pela economia de custos e pelo tempo para se alcançar um objetivo.

Segundo CODENIE, STEYAERT e VERCAMMEN (1997), o desenvolvimento de um *Framework* envolve muitas mudanças, tanto do ponto de vista técnico como gerencial. É preciso ter equipes com perfis diferentes para se formar uma equipe responsável pelo Ambiente, e outra equipe responsável pelas aplicações que irão derivar desse Ambiente. A equipe responsável pelo Ambiente tem como função efetuar a análise do domínio, construir soluções reusáveis, visto que suas atividades envolvem múltiplos projetos. A equipe responsável pelas aplicações está voltada para customizar o Ambiente, preenchendo e criando as funcionalidades específicas para completá-lo, o que gera uma aplicação. As atividades dos desenvolvedores de aplicação derivadas de um Ambiente diferem daquelas dos desenvolvedores de aplicação tradicional, visto que os primeiros nunca desenvolvem uma aplicação do zero, pois eles devem customizar uma solução já existente, o Ambiente.

A documentação é um dos pontos mais negligenciados pelos desenvolvedores de *software*, mas é muito importante para uma solução que usa um Ambiente. Fica muito mais difícil para os desenvolvedores utilizarem o Ambiente sem uma documentação adequada, o que os faz correrem o risco de não conseguir resultados de qualidade, além do fato de ser necessário muito mais tempo para seu entendimento e conseqüentemente para seu desenvolvimento.

O uso de padrões estabelecidos pode tornar o Ambiente mais fácil de ser entendido e mais robusto. Uma das vantagens de usar um padrão no processo de modelagem da solução é que eles podem ser usados na documentação também. Uma linguagem padrão deve ser usada para descrever um Ambiente equalizando os termos do negócio e da solução. O padrão da linguagem é importante, pois o Ambiente será utilizado por diferentes grupos de usuários, uma vez que atende vários produtos e que originalmente podem utilizar linguagens diferentes.

### 2.5.3 – Contribuições para o trabalho

A criação de um Ambiente permite o reuso, desde um nível abstrato como o de conhecimento, até um nível concreto de um componente de *software*. Mas o maior ganho do Ambiente é o reuso do *Framework*, visto que o desenvolvimento de uma nova aplicação já parte de uma solução pré-existente que deve ser complementada com as particularidades da nova aplicação. Os conceitos de Ambiente apresentados são utilizados na fase de Definição do Ambiente e na de Elaboração do Ambiente.

## 2.6 – Família de produtos ou Linha de produto de *software*

Os dois termos, família de produtos e linha de produto, são utilizados e carregam a mesma intenção. Nesse trabalho adota-se o termo família de produtos.

### 2.6.1 – Definições

Segundo SOMMERVILLE (2001), uma família de produtos é um conjunto de aplicações que apresenta uma arquitetura de domínio comum, porém cada aplicação contém sua especialização. O núcleo comum da família de produtos é reusado cada vez que uma nova aplicação é construída, trazendo, porém, alguns componentes adicionais e adaptando outros componentes para atender às novas demandas.

Para NIEMELÃ e IHME (2001), a idéia principal para a engenharia de *software* para uma família de produtos é o desenvolvimento de uma infra-estrutura de reuso que suporta o desenvolvimento de *software* para uma família de produtos. A análise de casos de uso e o escopo da família de produtos são os primeiros passos para o desenvolvimento da família de produto. Essa etapa vai estimar os benefícios do desenvolvimento da família de produtos, a carteira de produtos que são atendidos, os domínios que apresentam potencial para reuso, e os artefatos de *software* que são considerados como bens reusáveis da família de produtos.

### 2.6.2 – Análise de Domínio

Segundo COMER (1990), Análise de Domínio é um dos fundamentos para o processo de desenvolvimento de *software* voltado para reuso. É uma atividade da engenharia de

sistemas focada em uma família de sistemas de um determinado domínio de aplicações que visa ao desenvolvimento de bens reusáveis. Estão incluídos nesses bens componentes de *software*, documentações, especificações de interfaces, plano de testes, procedimentos e informações.

Para PIETRO-DIAZ (1990), Análise de Domínio é um processo no qual as informações relacionadas a um sistema em desenvolvimento são identificadas, capturadas e organizadas com a intenção de serem reusadas para novos sistemas.

O reuso deve ser considerado em um nível mais abstrato e abrangente que no nível de código. A reutilização deve ser acontecer por meio de artefatos que permitam o reuso de conhecimento, dos itens de solução de projeto e da infra-estrutura de domínio. Visando a esse objetivo, a Análise de Domínio vem ganhando reconhecimento nas últimas décadas, visto que tende a facilitar o reuso no processo de desenvolvimento.

A Análise de Domínio delimita o escopo de uma família de produtos, definindo quais sistemas estão dentro ou fora dessa família. O escopo é um fator de sucesso para um projeto que implanta uma família de produtos, pois um escopo muito estreito pode não justificar o investimento no seu desenvolvimento, enquanto que um escopo muito amplo pode levar a um esforço muito grande para desenvolver produtos individuais de forma econômica e reduzida.

É essencial, na construção de uma família de produtos, que esteja identificada a parte que será constante e comum a todos os membros dessa família e à parte variável. O problema do escopo não é encontrar as características comuns, mas encontrá-las de forma que possam ser exploradas para que a redução de custo de construção de sistemas realmente aconteça.

Segundo NIEMELÃ e IHME (2001), uma das maiores dificuldades de desenvolvimento de uma família de produtos é análise de domínio de conhecimento: qual o tipo de domínio de conhecimento que existe, como ele está representado e qual é a sua qualidade. O desenvolvimento de uma família de produto objetiva o reuso de requisitos de *software,* de arquitetura, de componentes e de processos. A análise de pontos em comum e de pontos variáveis das funcionalidades e das propriedades estruturais do sistema é a principal diferença entre o desenvolvimento de uma família de produtos e o desenvolvimento de um sistema de *software* simples.

### 2.6.3 – Arquitetura para Família de Produtos

A essência da construção de uma solução sistêmica para uma família de produtos, com sucesso, é a definição da parte que é esperada que permaneça constante para todos os membros da família, e daquela que é variável. Em uma solução para esse escopo, a arquitetura é a expressão dos aspectos constantes da solução.

O arquiteto de uma solução sistêmica para uma família de produtos deve considerar três aspectos:

- Identificação de pontos variáveis;
- Suporte a pontos variáveis;
- Definição da arquitetura adequada para a solução.

A arquitetura de uma família de produtos deve considerar e atender aos pontos variáveis do escopo, visto que uma solução mal dada para esses pontos variáveis pode colocar em risco a evolução dessa família de produtos. Caso a implementação de um novo produto necessite de muito desenvolvimento ou de muitas adequações na parte que foi considerada constante, pode haver descaracterização dos conceitos básicos propostos, o que faz perder as vantagens desse tipo de solução.

NIEMELÃ e IHME (2001) apresentam os seguintes elementos de análise para a arquitetura de uma família de produtos:

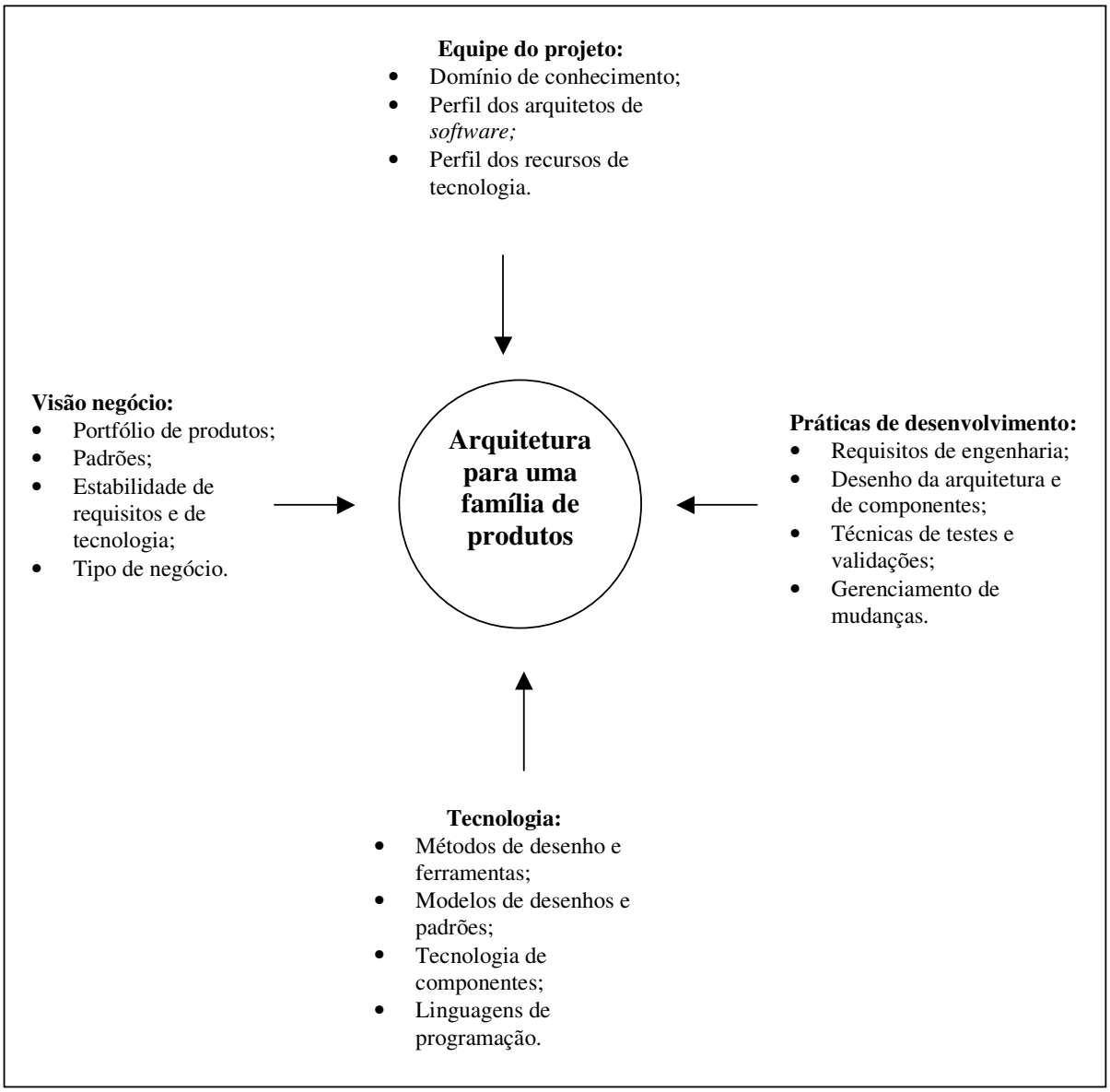



Figura 3 – Elementos de análise da arquitetura de família de produtos (NIEMELÃ e IHME, 2001).

Segundo NIEMELÃ e IHME (2001), se a empresa tem uma visão do tipo de características dos produtos que são necessárias naquele momento, quais irão mudar e das novas características que serão necessárias no futuro, ela permite que o mapeamento do negócio esteja bem conectado com a estratégia de tecnologia e de desenvolvimento de *software*, o que é um grande fator de sucesso para a solução a ser dada para a família de produtos. As empresas que tiveram maior sucesso na solução para uma família de produtos,

foram as que melhor tinham conhecimento de domínio e de arquitetura de desenvolvimento, e também apresentavam uma estrutura organizacional mais flexível.

### 2.6.4 – Contribuições para o trabalho

A criação de uma solução para uma família de produtos é um grande objetivo desse trabalho. Uma vez definida a viabilidade da criação da família de produtos, a solução a ser dada tende a ter um alto nível de reuso, o que gera aplicações com qualidade, custo e prazos mais baixos, à medida que vai se tornando mais estável. Os conceitos definidos nesse item são aplicados na fase de Elaboração da Análise de Domínio e Definição da Arquitetura da Solução.

## 2.7 – Conclusão

Os assuntos apresentados nesse capítulo fornecem fundamentação teórica e base de conhecimento para a elaboração e entendimento do roteiro a ser apresentado no capítulo 3.

O roteiro apresentado visa a propiciar o desenvolvimento de *software* com qualidade, menor prazo e menor custo de desenvolvimento. Para isso, os conceitos de qualidade de *software* são previamente definidos e, juntamente com a aplicação de técnicas de arquitetura de *software* e reuso, espera-se alcançar os objetivos propostos.

# Capítulo 3 - Roteiro para a Criação de um Ambiente Baseado em Componentes para Família de Produtos

Nesse capítulo, é apresentado o roteiro proposto para o desenvolvimento de um Ambiente baseado em componentes para uma família de produtos. Cada fase do roteiro é descrita, detalhando suas atividades e destacando, ao final de cada uma delas, o tipo de artefato de reuso possível de ser utilizado ou gerado.

## 3.1 – Fases do Roteiro

O roteiro proposto nesse trabalho teve como referência o modelo Clássico de desenvolvimento de *software,* que contempla a execução seqüencial das seguintes fases:

- Levantamento de requisitos;
- Análise;
- Projeto;
- Codifição;
- Teste.

Após a entrega da primeira versão do sistema, ao ocorrer necessidade de evolução, mudanças ou correções, novas versões são geradas, o que caracteriza a fase de Manutenção.

O modelo Clássico foi usado como referência por ser um modelo que contempla as atividades básicas para o desenvolvimento de *software* na sua forma mais simples e elementar, e que ainda é largamente utilizado.

Com as pesquisas efetuadas sobre os assuntos apresentados no capítulo 2 que fornecem o embasamento teórico para o trabalho, algumas fases foram adaptadas, outras foram incluídas para que as novas necessidades para a criação do roteiro fossem atendidas.

O roteiro proposto nesse trabalho apresenta as seguintes fases, conforme ilustrado na Figura a seguir:

Fase 1: Elaboração da Análise do Domínio;

Fase 2: Definição da Arquitetura da Solução;

Fase 3: Definição do Ambiente;

Fase 4: Elaboração dos Componentes;

Fase 5: Elaboração do Ambiente;

Fase 6: Desenvolvimento da Aplicação.

As fases 1 a 5 podem ser consideradas fases de *set up* da família de produto. Ao final dessas fases, o Ambiente para atender à família de produtos está pronto e deve sofrer poucas manutenções, pois corresponde à parte que é comum e constante para essa família de produtos.

A fase 6 tem como característica a repetição de geração de uma aplicação derivada do Ambiente. Corresponde às especializações e, portanto, à parte variável da solução que atende às necessidades particulares do produto da família que a aplicação deve suportar.

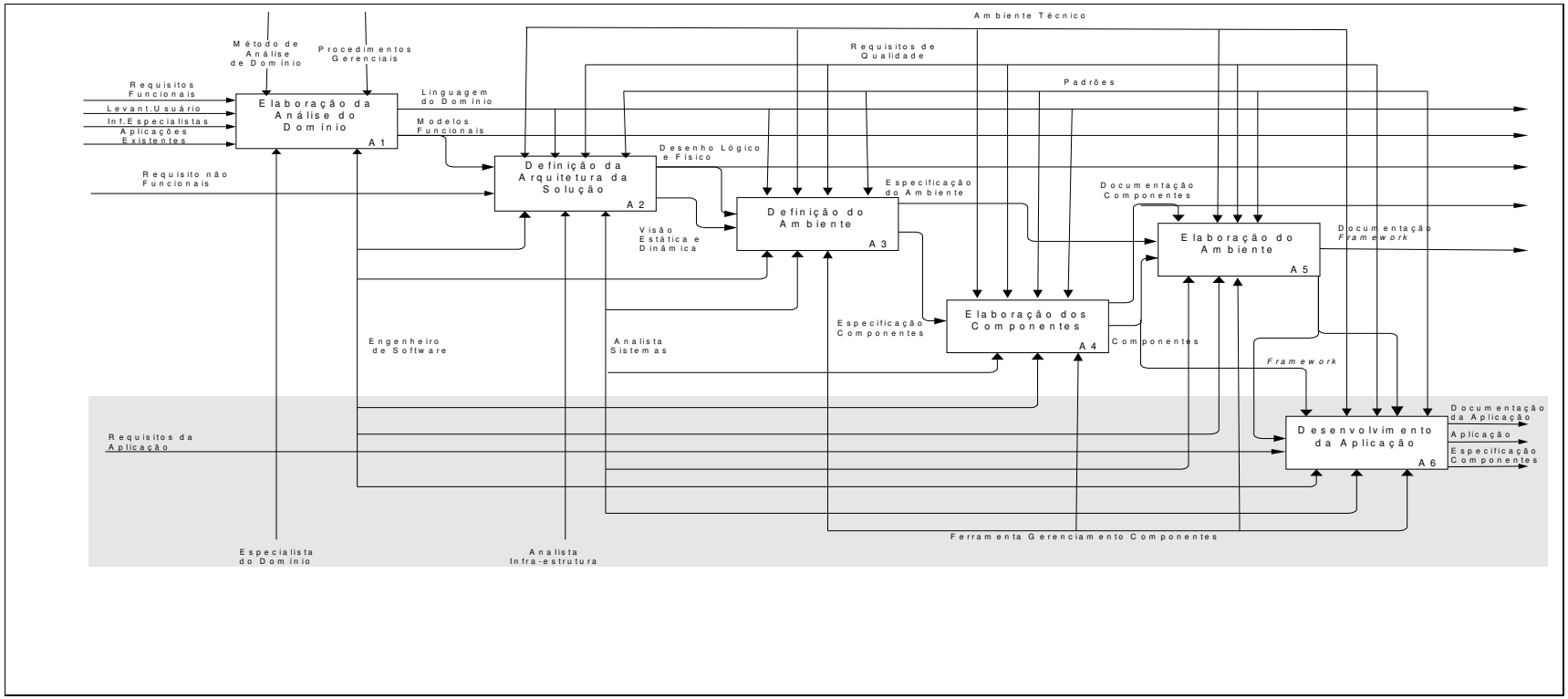



Figura 4 – Roteiro proposto.

As atividades do modelo Clássico são todas utilizadas no roteiro proposto, pois como já citado anteriormente, elas são básicas para o processo de desenvolvimento de *software*. A diferença no roteiro proposto é que elas são executadas mais de uma vez em contextos diferentes. A atividade de manuntenção não é detalhada no roteiro proposto, mas após a implantação, tanto o Ambiente como as aplicações podem sofrer manutenções, passando pelo ciclo de vida natural de um sistema.

### 3.2 – Fase 1: Elaboração da Análise do Domínio

Essa fase contempla as atividades de levantamento dos requisitos e de análise do modelo Clássico, na qual o grande objetivo é definir os requisitos do projeto que vão atender à família de produtos em questão, principalmente estabelecendo os requisitos que são constantes e comuns e os que são variáveis para essa família.

A análise de domínio pode ser considerada equivalente ao levantamento convencional de requisitos. No entanto, ao atuar em um nível mais amplo, em vez de explorar requisitos de uma aplicação específica, os requisitos explorados dizem respeito a uma família de aplicações de uma determinada área.

A importância de análise de domínio no contexto de um processo de desenvolvimento *para* e *com* reuso é que, ao ser realizada a análise e a exploração do conhecimento envolvido na aplicação e em aplicação similares, os modelos gerados tendem a ser mais estáveis, as mudanças tendem a ser isoladas e as aplicações a se tornarem, por conseqüência, mais robustas.

A especialização em uma determinada família de produtos exige a definição clara de um escopo para que seja possível a implantação de novos produtos nessa solução sem deturpar ou ferir a arquitetura original. A entrada de um novo produto deve ocorrer de forma natural, caso contrário uma análise criteriosa deve ser realizada e, nesse caso, ou o produto não pertence a essa família, ou a solução necessita de ter seu escopo reavaliado e, conseqüentemente, todos os processos posteriores do roteiro.

Essa atividade é complexa sendo, portanto, necessário um processo bem definido e estruturado para o seu desenvolvimento. A análise de domínio toma a forma de conjunto interdisciplinar o qual envolve adaptações de técnicas comumente usadas nas áreas de engenharia de *software*, de engenharia de requisitos, de modelagem conceitual, de aquisição e de representação de conhecimento.

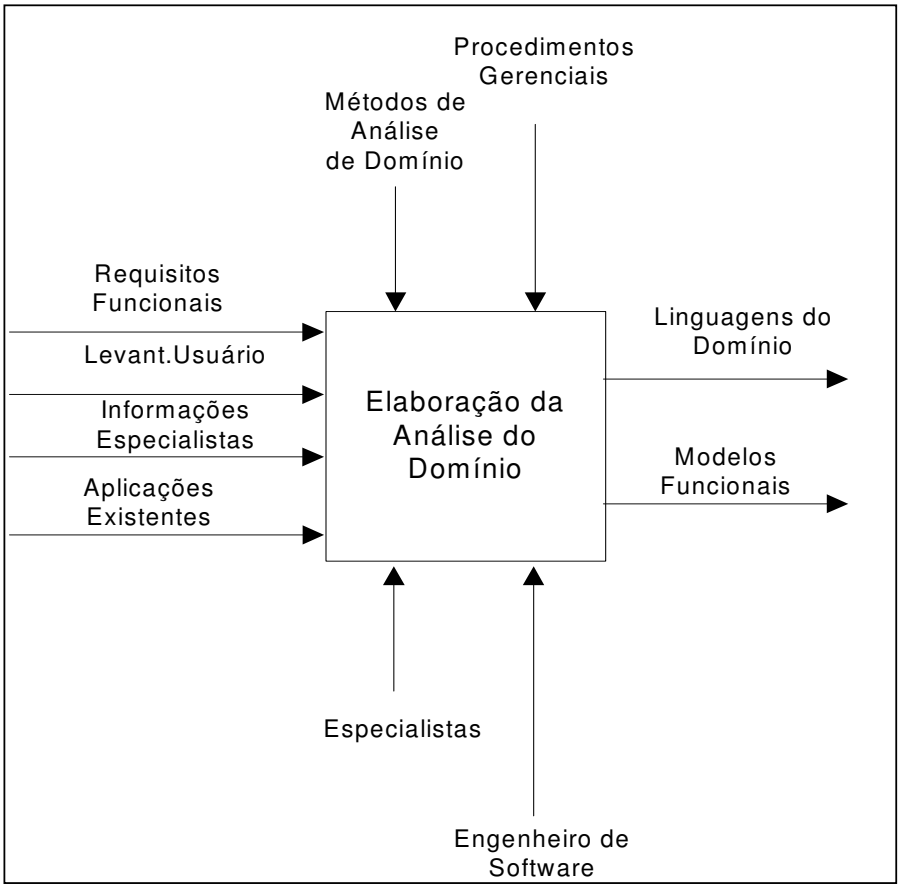



Figura 5 – Modelo da "Elaboração da Análise do Domínio".

Para essa análise, a coleta de informações é muito importante e pode ser obtida das mais diversas formas. Primordialmente, devem-se coletar informações dos especialistas e dos usuários dos produtos a que se quer atender e também de sistemas legados que porventura já existam.

De forma geral, o modelo de domínio é composto por um modelo conceitual e por um léxico (dicionário). Esse último visa de definir o padrão de linguagem que será adotada pela solução. Os modelos não devem, ainda, refletir decisões de implementações. Inicialmente deve ser mantido um escopo amplo. Como eles descrevem negócios, devem ser gerados modelos de tipos estáticos para representar os vocabulários de conceitos envolvidos, dos modelos de colaborações, dos casos de uso para representar os processos, as tarefas de pessoas, os departamentos e as regras envolvidas.

Nessa etapa, devem ser considerados os sistemas com os quais a família de produtos deve se relacionar, recebendo informações ou fornecendo informações.

As modalidades de reuso utilizados ou gerados nessa fase são:

Tabela 1 – Modalidades de reuso da fase 1: Elaboração da Análise do Domínio.

| **Tipo de Reuso** | **Descrição** |
|---|---|
| *Conhecimento* | É praticada por meio da documentação sistemática do conhecimento a respeito dos produtos em análise e de todos os processos envolvidos. O simples registro das informações não é o suficiente, a informação deve ser especificada de uma forma fácil de ser encontrada, usada e modificada. |
| *Linguagens/ Terminologia* | Com o envolvimento de vários produtos e, conseqüentemente, de várias áreas gestoras/usuárias que utilizam terminologias diferentes, a definição e padronização das terminologias mais adequadas a serem utilizadas é fundamental para o desenvolvimento e entendimento do projeto. |
| *Documentação* | Documentações de aplicações já existentes e com afinidade com os assuntos em questão são importantes fontes de conhecimento e também de informações para a construção de nova solução. |

### 3.3 – Fase 2: Definição da Arquitetura da Solução

Essa fase corresponde a uma das atividades da fase de projeto do modelo Clássico. A definição da arquitetura fornece a fundamentação técnica para uma estratégia de solução que permite o desenvolvimento de sistemas sobre uma infra-estrutura sólida e integrada, sendo que tal arquitetura é constantemente incrementada por novos elementos sem causar abalos em sua estrutura. Segundo KRUCHTEN (1995), a arquitetura de *software* se refere à abstração, com composição e decomposição, com estilo e estética.

À medida que os sistemas crescem, cresce também sua complexidade o que torna mais difícil satisfazer um número cada vez maior de requisitos, bem como atender às restrições de orçamento e do cronograma. Numa visão e na estratégia de reuso de *software*, tem-se enfatizado a importância de reuso centrado na arquitetura para o desenvolvimento de *software* durante todo o seu ciclo de vida. A arquitetura de *software* serve como uma estrutura que permite o entendimento de componentes de um sistema e seus inter-relacionamentos.

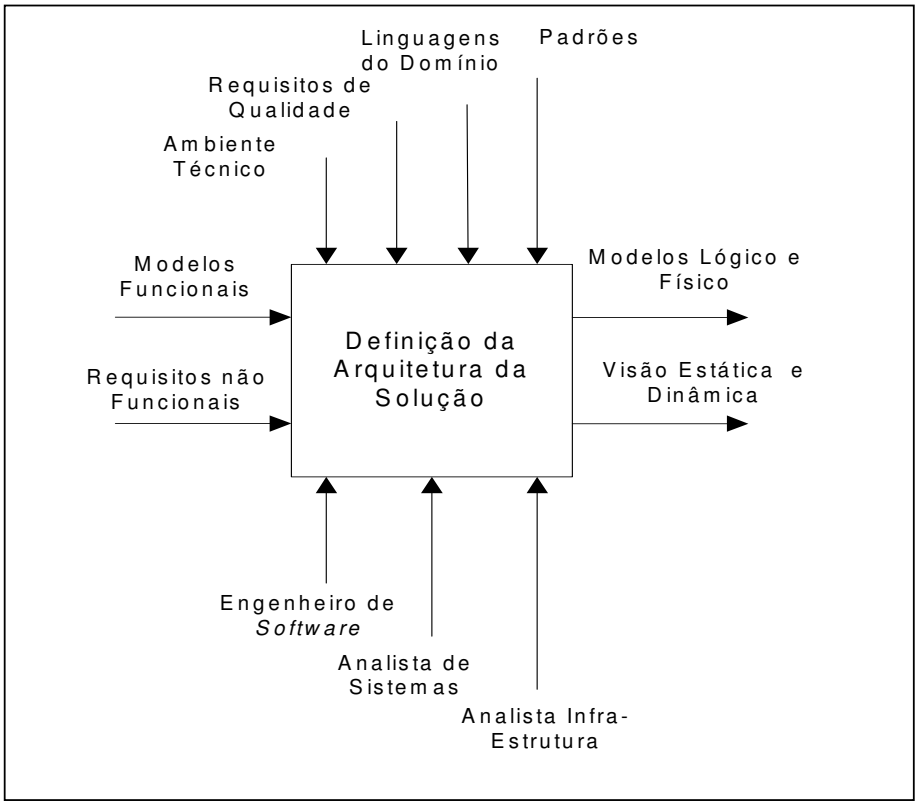



Figura 6 – Modelo da "Definição da Arquitetura da Solução".

Para a definição do projeto arquitetural, os requisitos funcionais e principalmente os não funcionais devem estar bem definidos e priorizados, visto serem as referências para uma série de definições em que conflitos podem surgir, tornando necessária a consideração de prioridades para a tomada de decisões na fase de projeto da solução.

Uma arquitetura de solução mal definida ou inadequada às reais necessidades de seus usuários traz conseqüências para toda a vida do *software* a ser gerado, haja vista que uma mudança arquitetural, após a construção do sistema, provavelmente requer muito tempo e orçamento o que, em casos mais graves, pode ser inviável.

A definição da arquitetura precisa atender, dentro do possível, aos requisitos de qualidade que a corporação venha a exigir, assim como acatar as disponibilidades e restrições do ambiente tecnológico que a empresa apresenta. Caso seja necessária a aquisição, criação ou mudança de infra-estrutura tecnológica para atender a uma arquitetura mais flexível e robusta, novamente será necessária uma análise e definição de prioridades, visto que tal ocorrência pode acarretar um aumento de custo para o projeto.

Nesses casos, uma visão corporativa é fundamental para que sejam adotadas soluções que sejam reutilizadas em outros projetos o que rateia seus custos e, principalmente, cria padrões de soluções.

As relações com os demais sistemas já existentes também devem ser consideradas e podem influenciar em algumas definições da arquitetura, como a criação de camadas específicas para atender a esses relacionamentos.

Essa etapa é de responsabilidade do engenheiro de *software* e dos analistas de infra-estrutura. Uma arquitetura bem montada, de modo lógico e físico, é fundamental para o sucesso do projeto. Deve-se partir do princípio que, em uma solução de arquitetura de *software,* deve ser utilizado o melhor da tecnologia para atender aos requisitos, mesmo que ao longo do processo se vá abrindo mão de algumas arquiteturas, após a análise de custo e benefício. Nessa etapa, a participação do Analista de Sistemas já pode ser iniciada para que ele obtenha familiaridade e conhecimento necessário para as próximas etapas em que sua participação é intensa.

Como resultado dessa fase, é gerada uma série de modelos os quais representam as várias visões da arquitetura definida para a solução. Uma visão importante na arquitetura de uma solução de um ambiente para uma família de produtos é a definição da parte fixa e da parte variável do projeto. A parte fixa representa aquela que é comum a todos os produtos e que, portanto, sofre reuso. A parte variável é a parte que atende às necessidades particulares de cada produto e que, provavelmente, tem que ser desenvolvida na implantação desse produto. Essa parte variável dá a flexibilidade à solução na implementação de novos produtos.

As modalidades de reuso utilizados ou gerados nessa fase são:

Tabela 2 – Modalidades de reuso da fase 2: Definição da Arquitetura da Solução.

| **Tipo de Reuso** | **Descrição** |
|---|---|
| *Modelos de Arquitetura* | Os modelos e diagramas que representam as várias visões da arquitetura podem ser aproveitados de soluções já existentes ou serem utilizados em soluções futuras, principalmente se tiverem similaridades de escopo e um histórico de sucesso. A tendência é que sistemas similares utilizem uma mesma arquitetura, o que viabiliza o reuso, inclusive de *hardware* e *middleware.* |
| *Idéias arquitetônicas* | Registro que um arquiteto faz de suas experiências pertinentes para discenir sobre o que adere bem e o que não adere bem como solução técnica e de negócios ao problema em questão. |
| *Estilos e padrões de arquitetura* | Os estilos e padrões de arquitetura consagrados na literatura, adequando por semelhança ao negócio e à representação técnica disponível, servem como primeira referência para a escolha da arquitetura. |

## 3.4 – Fase 3: Definição do Ambiente

Essa fase envolve atividades das fases de projeto e análise do modelo Clássico voltadas para a visão geral da solução que dá origem ao *framework* que atende à família de produtos.

O desenvolvimento de um Ambiente para uma família de produtos se torna viável quando atende a um número razoável de produtos, pelo menos três, e quando existem muitas funcionalidades em comum entre eles. Caso contrário, provavelmente, o custo e o benefício não são compensadores.

Para tanto, a análise de domínios realizada na etapa inicial do roteiro deve ser criteriosa e conclusiva na viabilidade e no retorno do desenvolvimento do Ambiente para a família de produtos em questão.

Um dos principais resultados dessa fase é o modelo do *framework* da solução a ser construída. O modelo de *framewok* é um pacote genérico que contém as definições comuns e as definições que são substituídas. É a estrutura de uma solução para uma aplicação que apresenta as partes fixas, as comuns e a parte que é substituída para atender às particularidades de uma aplicação que é desenvolvida a partir desse Ambiente.

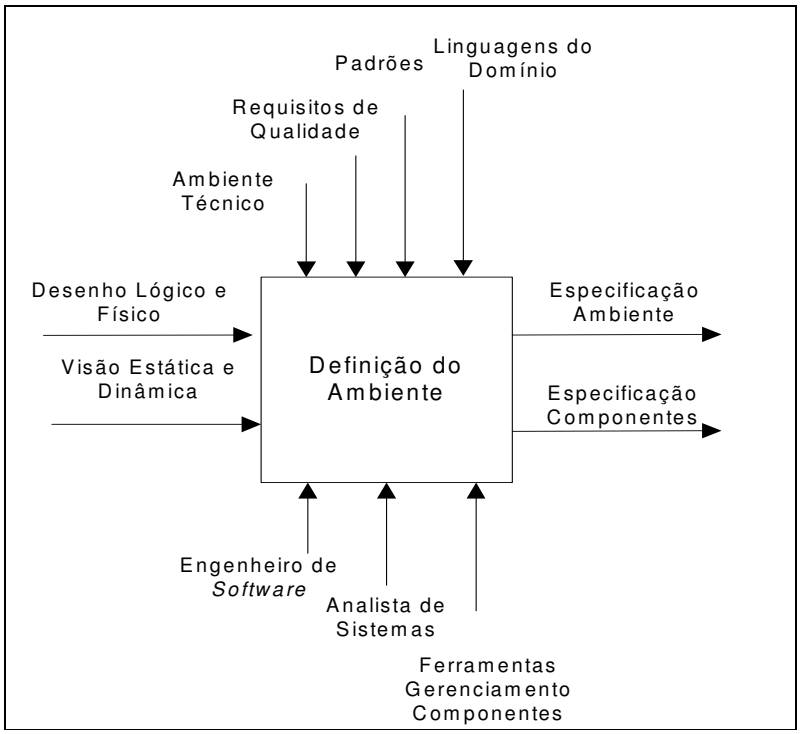



Figura 7 – Modelo da "Definição do Ambiente".

A partir dos modelos e diagramas lógicos e físicos e da visão estática e dinâmica, gerados na fase de elaboração de análise do domínio e definição da arquitetura da solução, o Ambiente deve ser projetado, detalhado e especificado.

O projeto do Ambiente deve mostrar, de forma abstrata, a estrutura que o *Framework* vai ter, apresentando todas as funcionalidades a que ele atende e já identificando as funcionalidades que são comuns e fixas e as que são variáveis. As partes variáveis devem ser

identificadas e reconhecidas pelo Ambiente por parametrizações criadas. Esse é um ponto crítico para o projeto e deve ser muito bem trabalhado e definido.

Uma vez definido o projeto do Ambiente, deve ser detalhada cada uma das fases do Ambiente e suas respectivas funcionalidades já visando à componentização do projeto. Deve-se ter sempre como escopo criar componentes simples e com objetivos claros.

Com os componentes necessários para o projeto definidos, deve ser realizada uma busca em componentes já existentes para reutilização. O objetivo do DBC é que, ao desenvolver um novo projeto, quanto mais componentes prontos são reutilizados melhor, porque o tempo para conclusão do projeto diminui e, conseqüentemente, o seu custo, bem como a qualidade é garantida pelo fato de os componentes reusados já estarem estabilizados e validados pelas utilizações anteriores.

Um dos fatores fundamentais que influenciam no sucesso da pesquisa de componentes são as informações cadastradas para identificar e descrever um componente. A pesquisa exige uma metodologia, pois nem sempre são utilizados os termos adotados para o projeto em questão, tanto que é preciso pesquisar por palavras similares e por palavras-chaves.

A ferramenta disponível para pesquisa dos componentes também é importante, sendo que deve oferecer mecanismos de pesquisa eficientes e de fácil utilização. Um trabalho atento e minucioso deve ser realizado pelos engenheiros de *software* nas bases de componentes disponíveis, pois quanto mais componentes necessários estiverem prontos, melhor.

Os componentes necessários para a criação do Ambiente que não são encontrados nas bases de dados de componentes precisam ser especificados. Um componente bem definido e projetado é crucial para resultar em um alto nível de seu reuso e na facilidade de manutenções que porventura vierem a ser necessárias. Portanto, essa fase exige muito conhecimento e experiência por parte dos engenheiros de *software* envolvidos.

É recomendável que seja adotado um padrão nas interfaces dos componentes e do Ambiente, de forma que o entendimento e a utilização deles sejam facilitados.

Como o Ambiente é utilizado por equipes e por profissionais diversos, isto é, por ele trazer o conceito de serviço a ser utilizado e, principalmente, por necessitar de partes de encaixe na implementação de um novo produto, é muito importante que as definições do Ambiente obedeçam a um padrão de desenvolvimento e principalmente de interface. Vale ressaltar a recomendação de que a equipe que mantém o Ambiente deve ser diferente das equipes que mantêm os produtos gerados pelo Ambiente, uma vez que os objetivos e até os

perfis dos profissionais dessas equipes são diferentes. O Ambiente não pode ser influenciado por características específicas de nenhum produto, visto que deve atender a todos, na medida do possível.

As modalidades de reuso utilizados ou gerados nessa fase são:

Tabela 3 – Modalidades de reuso da fase 3: Definição do Ambiente.

| **Tipo de Reuso** | **Descrição** |
|---|---|
| *Requisitos/ Funcionalidades* | O Ambiente é definido com base nos requisitos que estabelecem as fucionalidades que devem ser atendidas pela família de produtos. Na maioria das vezes, essas funcionalidades já são contempladas por sistemas existentes, podendo ser reaproveitados seus requisitos, seu modelo de solução ou até mesmo a própria solução. |
| *Definição de Componentes* | Um componente é inicialmente identificado com base na documentação que contém suas definições de negócios, portanto deve ser bem elaborada e estar disponível para seus possíveis usuários. Ela compõe uma das partes da documentação do componente. |
| *Documentação do Ambiente* | A documentação gerada referente ao Ambiente é utilizada para a implementação de qualquer aplicativo que vier a derivar desse Ambiente. |

## 3.5 – Fase 4: Elaboração dos Componentes

Essa fase envolve atividades das fases de codificação e testes do modelo Clássico dos componentes especificados.

Ao projetar um componente reusável, deve-se estar atento para que ele seja o mais simples possível, pois um componente muito complexo dificulta sua reutilização ou determina que apenas uma pequena porção do componente seja usada.

Segundo D´SOUZA e WILLS (1998), ao se construir um sistema usando componentes, é necessário um conjunto de padrões que devem ser de conhecimento e serem

seguidos pelos desenvolvedores para que esses componentes trabalhem de forma integrada e reduzam a carga de desenvolvimento.

Existem as seguintes categorias de padrões:

- *Padrões horizontais* – Infra-estrutura: os componentes precisam de mecanismos básicos de padrão para trabalhar com padrões de chamadas e de respostas, com segurança para autenticar seus usuários, com controle de transações para garantir a função de persistência, bem como com padrões de diretórios para acessar diretórios de serviços;
- *Padrões verticais* – Definição de termos utilizados no domínio do problema. As definições devem ser compartilhadas nas interfaces entre os componentes;
- *Padrões de conexão* – Definição da variedade de mecanismos de interação entre os componentes, por exemplo, se a chamada é síncrona ou assíncrona ou se mudanças de estados são propagadas para outros componentes.

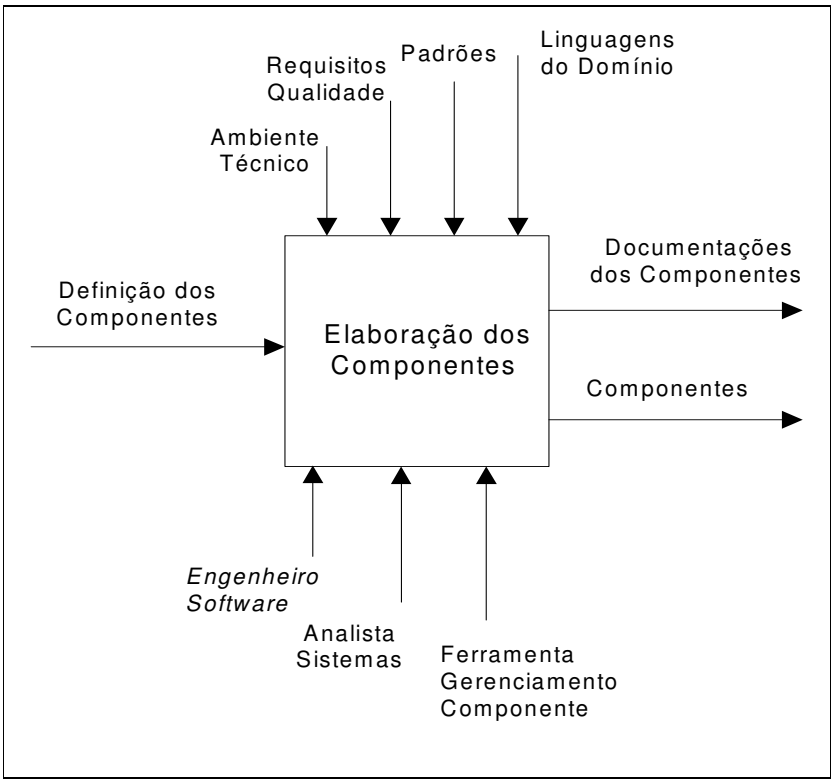



Figura 8 – Modelo da "Elaboração dos Componentes".

A componentização exige a definição de interfaces precisas para que os componentes possam se comunicar perfeitamente no momento em que forem utilizados.

Essa fase exige grande experiência e conhecimento dos engenheiros de *software* e dos analistas de sistemas envolvidos, pois somente componentes bem projetados têm alto nível de reuso e flexibilidade. Os atributos de qualidade referente ao desempenho e à portabilidade são importantes nessa fase.

A disponibilidade de uma documentação completa do componente aumenta sua qualidade, dando precisão e segurança para quem deseja utilizá-lo, o que contribui para o aumento da possibilidade de reuso.

As ferramentas de desenvolvimento e, principalmente, a linguagem a ser utilizada devem ter uma escolha criteriosa, visto que afetam diretamente a qualidade e a produtividade dos componentes gerados.

Uma empresa pode ter um grande acervo de componentes, porém se não tiver uma ferramenta que gerencie e permita uma recuperação fácil desses componentes, a implantação e a aplicação do DBC provavelmente se tornam bastante comprometidas.

Cada componente gerado e devidamente testado deve ser catalogado na ferramenta de Gestão de Componentes utilizada, para que conste na relação de componentes disponíveis para reuso com base na documentação fornecida pelo Engenheiro de *Software* e/ou Analistas de Sistemas responsáveis pelo componente.

O cadastramento de um novo componente deve seguir uma metodologia definida pela empresa, porque uma série de pré-requisitos deve ser atendida antes de sua publicação para que a qualidade e, conseqüentemente, a confiabilidade dos componentes disponíveis possam ser garantidas.

A figura de um gestor de componentes que zele por essa qualidade e pela confiabilidade dos componentes disponibilizados deve existir. Assim como a ferramenta que gerencia o acervo de componentes disponíveis deve ser bem escolhida, visto que ela é o meio de acesso e de venda dos componentes para os seus prováveis usuários.

É aconselhável que algumas informações sobre o componente sejam diponibilizadas, tais como: a data de disponibilização do componente, a equipe responsável, quais os programas/rotinas/sistemas que já o utilizam e desde quando o fazem, o histórico de

ocorrências desse componente, entre outras. Essas informações são importantes, porque possibilitam uma maior transparência para os usuários ou para os futuros usuários dos componentes, assim como um melhor gerenciamento deles.

A base de dados de componentes é um bem de grande valor para a empresa, visto que, além de gerir uma base de conhecimento dos negócios da empresa em forma de *software*, cujo desenvolvimento apresenta um custo atraente para a empresa, permite agilidade no desenvolvimento de novos *software*s, de forma rápida e segura por meio do reuso.

As modalidades de reuso utilizados ou gerados nessa fase são:

Tabela 4 – Modalidades de reuso da fase 4: Elaboração dos Componentes.

| **Tipo de Reuso** | **Descrição** |
|---|---|
| *Documentação dos Componentes* | A documentação gerada para cada componente criado é seu principal recurso de venda. Nessa etapa, é gerada a documentação técnica do componente. |
| *Componentes de Software* | São os artefatos gerados que são usados pelos demais sistemas e aplicações no momento da execução. |

### 3.6 – Fase 5: Elaboração do Ambiente

Essa fase envolve atividades das fases de codificação e testes do modelo Clássico para o *framework* a ser desenvolvido para atender à família de produtos.

Essa fase pode ser iniciada em paralelo com a fase de Elaboração de Componentes, caso haja recursos para tal, uma vez que seu projeto está definido e que os componentes necessários para seu desenvolvimento, ou já existem, ou estão sendo construídos conforme solicitação resultante da fase 3 – Definição do Ambiente.

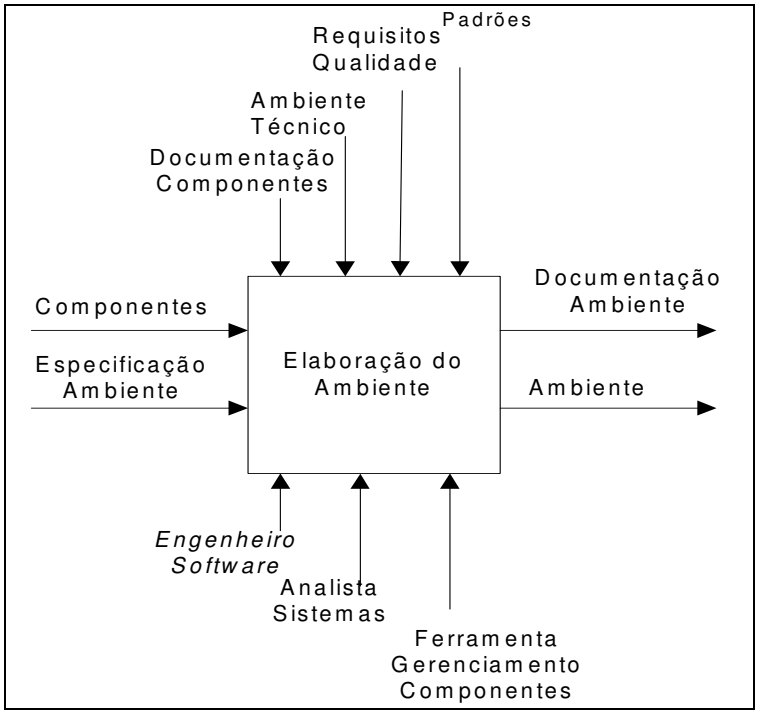



Figura 9 – Modelo da "Elaboração do Ambiente".

A partir das especificações geradas na fase de definição do Ambiente, são construídos os módulos que fazem parte dele, atendendo às funcionalidades definidas. Nessa fase, devem ser considerados os atributos de qualidade, de usabilidade e de eficiência do Ambiente, além de complementar a implementação física dos atributos de qualidade de desempenho e portabilidade definidos nas fases anteriores.

A escolha da linguagem que é utilizada na implementação é muito importante, porque pode afetar diretamente na qualidade, na produtividade dessa fase e principalmente na compatibilidade com componentes a serem utilizados e na interoperabilidade para os aplicativos a serem derivados.

A saída dessa fase é um Ambiente documentado, desenvolvido, testado e pronto para ser utilizado para a implementação de aplicativos que atendam os novos produtos da família de produtos.

Provavelmente o Ambiente será usado por equipes de desenvolvimento diferentes, portanto, para ser entendido e utilizado com toda sua potencialidade, precisa ter uma documentação clara, detalhada e atualizada.

As modalidades de reuso utilizadas ou geradas nessa fase são:

Tabela 5 – Modalidades de reuso da fase 5: Elaboração do Ambiente.

| **Tipo de Reuso** | **Descrição** |
|---|---|
| *Documentação do Ambiente* | A documentação do Ambiente é fundamental para a sua manutenção e evolução e, principalmente, para que, de forma rápida, eficiente e segura, sejam derivados aplicativos. |
| *Ambiente* | Esqueleto de aplicação a ser completado com as características específicas do produto a ser atendido, gerando novas aplicações. |
| *Componentes de software* | Reuso de partes codificadas do sistema. Essas partes precisam encapsular funcionalidades. Podem ter sido desenvolvidas na fase anterior, ou projetos anteriores ou serem adquiridas. |

## 3.7 – Fase 6: Desenvolvimento da Aplicação

Essa fase envolve atividades das fases de análise, codificação e testes do modelo Clássico, visando à criação de aplicações a partir do Ambiente gerado. A análise, codificação e testes são referentes às funcionalidades especialistas correspondentes à parte variável do Ambiente.

Essa fase é executada várias vezes, gerando sucessivas aplicações para os produtos da família em questão, onde as funcionalidades comuns e constantes serão reusadas, gerando, dessa forma, uma aplicação de forma rápida e conseqüentemente com menor custo.

Ao surgir a necessidade de desenvolvimento de uma aplicação para atender a um novo produto da família já tratada pelo Ambiente, deve ser analisado se esse novo produto, por suas características, pode ser atendido por uma aplicação já existente ou se é necessário o desenvolvimento de uma nova aplicação a partir do Ambiente.

Caso se conclua que deve ser desenvolvida uma nova aplicação a partir do Ambiente, os passos para tal são detalhados a seguir.

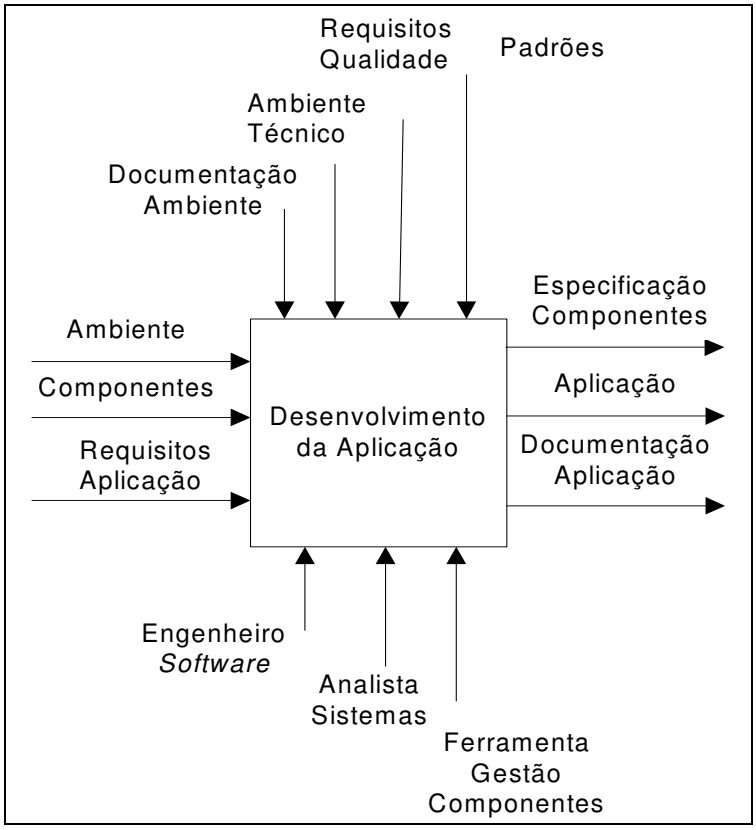



Figura 10 - Modelo do "Desenvolvimento da Aplicação".

Primeiramente, é necessário verificar se o produto a ser trabalhado realmente participa da família de produtos a que o Ambiente trata. Para isso, a documentação do Ambiente deve ter um *check-list* das características operacionais e funcionais a que ele atende.

Uma vez verificado que o produto pode ser tratado pelo Ambiente, deve ser efetuado um levantamento e gerada uma especificação das suas particularidades para que sejam desenvolvidas e/ou adaptadas para sua implementação. Essa especificação deve ser baseada na documentação do Ambiente e dos componentes que porventura vier a ser utilizada para a nova aplicação. Caso haja a necessidade de criação de novos componentes, os mesmos procedimentos da fase 4 – Elaboração dos Componentes devem ser seguidas.

Nessa etapa, pode ser necessária alguma adaptação no Ambiente, principalmente na implantação dos primeiros aplicativos. Essas adaptações devem ser realizadas sem impactar as aplicações derivadas do Ambiente que já tenham sido desenvolvidas. E espera-se que, com o uso, esse Ambiente se estabilize. Quanto maior a qualidade de arquitetura do Ambiente

mais rapidamente o Ambiente se estabiliza e mais facilmente as adaptações necessárias são realizadas.

O desenvolvimento da aplicação também envolve o trabalho do Engenheiro de *Software* e do Analista de Sistemas, apoiados pela Ferramenta de Gestão de Componentes para eventuais consultas à base de componentes, visto que o processo do DBC é um processo contínuo.

A saída dessa fase é uma nova aplicação que atende a um novo produto, assim como a documentação da referida aplicação.

As modalidades de reuso utilizados ou gerados nessa fase são:

Tabela 6 - Modalidades de reuso da fase 6: Desenvolvimento da Aplicação.

| **Tipo de Reuso** | **Descrição** |
|---|---|
| *Documentação da Aplicação* | A documentação da aplicação deve detalhar suas funcionalidades específicas. Ela será utilizada na avaliação da implementação de futuros produtos, auxiliando na decisão se são incorporados a aplicações já existentes ou se é necessária a criação de novas aplicações para atendê-los. |
| *Documentação dos Componentes* | A documentação gerada para cada componente criado é seu principal recurso de venda. Nessa etapa é gerada a documentação técnica do componente. |
| *Componentes de software* | Reuso de partes codificadas do sistema. Essas partes precisam encapsular funcionalidades que podem ser desenvolvidas ou adquiridas. |
| *Documentação do Ambiente* | A documentação do Ambiente é útil para definir se o novo produto pode fazer parte dessa família e, uma vez definido que sim, é utilizada para entender a solução de forma geral e na especificação das complementações necessárias para especializar a nova aplicação a ser desenvolvida. |
| *Ambiente* | O reuso do Ambiente se dá a cada nova aplicação gerada a partir dele. |

# Capítulo 4 – Exemplo de Aplicação do Roteiro

Esse capítulo visa usar, como exemplo, o roteiro apresentado no capítulo 3 para o desenvolvimento de um Ambiente para uma família de produtos de crédito bancário.

O objetivo desse exemplo da aplicação do roteiro é a sua validação e verificação para garantir que ele fique simples e completo o suficiente para ser de facilmente entendido e aplicado. É válido ressaltar que o foco dado no roteiro proposto é na visão do negócio e da modelagem da solução e não nas soluções técnicas. Um dos objetivos do trabalho é evidenciar que a modelagem de uma solução não deve ser influenciada pela plataforma tecnológica que é usada na sua construção, e sim por um enfoque do negócio, da necessidade a ser atendida, baseada em princípios de qualidade a serem alcançados.

Uma solução com um projeto e arquitetura bem definidos deve suportar a implementação em diferentes plataformas tecnológicas, sendo necessárias mudanças mínimas de adequação à plataforma e ao ambiente tecnológico escolhidos para sua implementação.

## 4.1 – Elaboração da Análise do Domínio

Devido à grande competição existente entre instituições bancárias, surge a necessidade de lançamentos de novos produtos com diferenciais cada vez maiores e com prazos cada vez menores. Para isso, os bancos investem fortemente em tecnologias que permitam soluções rápidas e inovadoras que ofereçam comodidades, modernidades e, principalmente, segurança aos clientes. A área bancária é uma grande investidora e consumidora de serviços na área de tecnologia.

Muitos dos sistemas de bancos são sistemas legados, que foram construídos há décadas e que vieram sofrendo atualizações e adaptações tanto por necessidades de negócios como de mudanças tecnológicas. Essas atualizações e adaptações, na maioria das vezes, não foram acompanhadas por mudanças estruturais nos sistemas, que permitissem uma solução flexível e que utilizassem da melhor forma as facilidades e potencialidades das novas tecnologias agregadas à solução.

Os sistemas nasceram e cresceram sem uma visão de solução e de arquitetura geral, com cada equipe cuidando de um sistema de um produto específico, o que resultou em um alto nível de redundância de informações e rotinas. Com a evolução dos negócios, há a necessidade de uma grande integração das informações entre esses sistemas e com sistemas

externos aos bancos, sistemas esses que utilizam as mais variadas tecnologias. Nesse cenário, a manutenção e a evolução dos sistemas tornam-se, na maioria das vezes, difíceis e com custos e prazos elevados.

Para solucionar essa situação, surge a necessidade de redesenhar e reconstruir esses sistemas legados, utilizando técnicas e conceitos mais modernos de solução sistêmica para melhorar sua qualidade de forma que os velhos problemas não voltem a se repetir. A solução baseada em componentes e o desenvolvimento de família de produtos são técnicas que auxiliam na obtenção desses objetivos.

Um banco comercial de varejo oferece uma gama de produtos de crédito bastante extensa para atender às necessidades de seus clientes. Pelas características desses produtos, existem muitas funcionalidades comuns entre eles na sua operacionalização e controle. E na maioria dos bancos, existem vários sistemas aplicativos para atender a esses produtos.

O escopo completo de um processo para operacionalizar um produto de crédito é bastante amplo e complexo. De uma forma geral, o processo de uma operação de crédito se inicia com a análise para a aprovação de limite de crédito para o cliente, passa pela concessão e controle de uma operação e é encerrado com a liquidação da operação ou no controle e cobrança de crédito vencido, como mostra a figura abaixo.

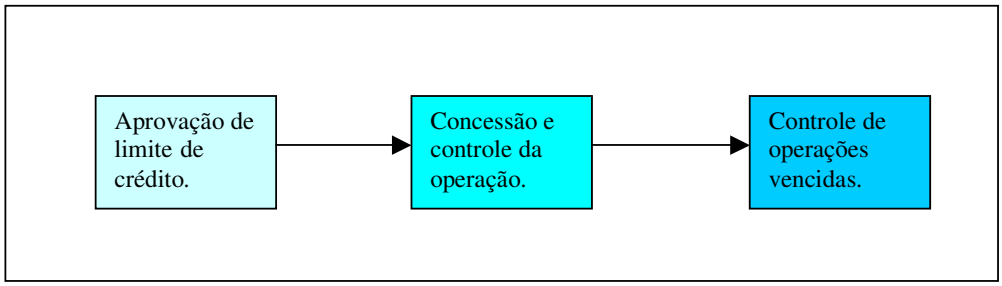



Figura 11 – Processo de uma operação de crédito.

Os processos de aprovação de limite e de controle de operações vencidas são comuns para qualquer produto de crédito e apresentam complexidade suficiente para que sejam tratados como sistemas específicos. Em virtude de tal fato, o escopo do exemplo da aplicação do roteiro é o processo de concessão e controle da operação de crédito.

Após os levantamentos com especialistas no negócio da área de crédito e com profissionais de tecnologia com experiência em sistemas aplicativos que suportam produtos de créditos, foram definidos os principais Casos de Uso, como um dos Modelos Funcionais resultantes dessa fase, conforme Tabela 7 a seguir:

Tabela 7 - Casos de uso.

| Ator | Nome Caso de Uso | Descrição |
|---|---|---|
| • Analista de Sistemas<br>• Gestor do Produto | Parametrização da aplicação | Ao se criar uma aplicação a partir do Ambiente, é necessário parametrizar a aplicação conforme suas necessidades. A parametrização define as funcionalidades do Ambiente a serem utilizadas e caracteriza outras conforme o produto a ser atendido. |
| • Cliente<br>• Funcionário | Solicitação da operação de crédito | O cliente solicita a operação de crédito informando a modalidade desejada e o valor. A solicitação da operação pode ser feita para um funcionário do banco ou, dependendo da modalidade, pode ser realizada diretamente pelo cliente no auto-atendimento ou pela Internet se já for cliente previamente cadastrado. A garantia para a operação de crédito também é informada, dependendo da modalidade. |
| • Funcionário | Avaliação da solicitação da operação de crédito | O(s) funcionário(s) designado(s) pelo banco avalia(m) a solicitação, a qual pode ser aprovada ou não, com base no Limite de Crédito concedido anteriormente para o cliente e nos riscos da operação. |
| • Funcionário | Registro e formalização da operação aprovada | Para as solicitações aprovadas, é realizado o registro da operação, com a complementação de informações, o que formaliza um contrato de crédito. |
| • Cliente<br>• Funcionário | Movimentação da operação de crédito | Com base no contrato gerado, são realizadas liberações de valores ao cliente, e esse, por sua vez, efetua as amortizações e liquidações das parcelas acordadas. |

| | | |
|---|---|---|
| • Funcionário | Aditivo da operação | Caso o cliente necessite, pode ser acordada uma mudança nas condições do contrato da operação, o que gera um aditivo de contrato. |
| • Cliente<br>• Funcionário | Extrato do contrato | O banco gera uma posição do contrato da operação, com uma determinada periodicidade ou a qualquer momento em que o funcionário e/ou o cliente solicitar essa posição. Essa consulta pode ser realizada em terminais internos do banco ou pela Internet, sempre com a opção de ser impressa. |
| • Sistema | Cálculo de juros | O cálculo dos juros pode ser pré ou pós-fixado, o que deve ser definido na contratação da operação. Os juros são calculados automaticamente pelo sistema nos períodos contratado conforme o índice de reajuste contratado. Para isso, são obtidas informações dos sistemas de índices financeiros e dos feriados para calcular dias úteis. |
| • Sistema | Cálculo de IOF | Deve ser utilizado o serviço corporativo já existente. |
| • Sistema | Cálculo de tarifas | Devem ser calculadas conforme a tabela definida pela área de produtos. |
| • Sistema | Cálculo de Comissões | Devem ser calculadas conforme a tabela definida pela área de produtos. |
| • Sistema | Integração com sistema de Cadastro de Clientes | Para que o cliente possa solicitar uma operação de crédito, ele deve estar cadastrado no sistema corporativo de cadastro de clientes, no qual constem informações pessoais, de domicílio e de renda. |
| • Sistema | Integração com sistema de Limite de Crédito | Para que uma solicitação de operação de crédito seja aprovada, o cliente precisa ter um limite disponível no sistema de Limites, para a modalidade solicitada de Crédito. |
| • Sistema | Integração com | Obedecendo aos prazos definidos e parametrizados, o |

| | | |
|---|---|---|
| | sistema de Créditos vencidos | sistema deve enviar as operações vencidas para o sistema de Créditos Vencidos para a emissão de avisos de cobrança, para controle de créditos em atraso e para controle de créditos em liquidação e prejuízo. |
| • Sistema | Integração com Sistema de Conta-Corrente | A liberação do valor da operação pode ser creditada diretamente na conta-corrente do cliente, assim como as amortizações são passíveis de serem debitadas da mesma forma. |
| • Sistema | Integração com sistema de Informações Gerenciais | Para controle e gestão interna e externa dos produtos de crédito, as informações das posições da carteira dos produtos devem ser passadas aos sistemas gerenciais conforme padrão e periodicidade definidos por eles. |
| • Sistema | Integração com sistema de Contabilidade | Toda operação, bem como toda movimentação da operação de crédito, necessita ser contabilizada; para tanto, deve ser construída uma interface com o sistema de Contabilidade. |
| • Sistema | Integração com cadastro de Agência | Toda operação de crédito precisa estar vinculada a uma agência reconhecida pelo Cadastro de Agências do Banco. |

Os Casos de Uso definidos na tabela acima exemplificam as funcionalidades básicas que a família de produto deve atender. Alguns Casos de Uso são opcionais e podem não ser usados para determinadas aplicações, como veremos mais à frente. Para tanto, o Ambiente a ser criado deve ser parametrizado, indicando qual funcionalidade será ou não utilizada em cada aplicação derivada.

Com base nos Casos de Uso definidos, o Diagrama de Contexto do Ambiente para uma família de produtos de crédito bancário a ser criado é o seguinte:

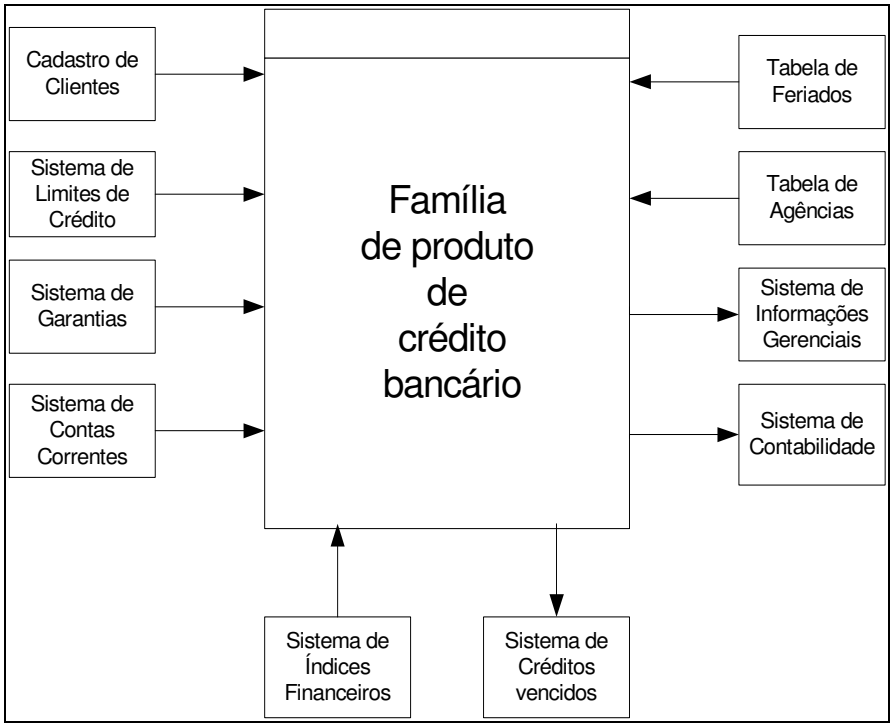



Figura 12 – Diagrama de contexto.

Para que uma operação de Crédito Bancário seja realizada, as pessoas envolvidas na operação, os tomadores e/ou os avalistas da operação necessitam ser conhecidos pela Instituição Bancária, por meio de um cadastramento de suas informações no **Cadastro de Clientes** do banco. Com base nas informações cadastrais e nas informações históricas de comportamento do cliente, é fornecido um **Limite de Crédito** para ele. Dentro do limite concedido ao cliente, é que a operação de crédito é realizada.

Na maioria das operações de crédito, porém não obrigatoriamente, o cliente necessita ter uma **Conta-Corrente** no banco, onde são efetuados os créditos e débitos dos valores envolvidos na operação. As garantias oferecidas na operação são validadas e ficam registradas no **Sistema de Garantias**.

Uma operação de crédito precisa estar vinculada a uma **Agência** da rede bancária para que seja gerida por ela. E para efetuar os cálculos durante a vigência da operação, o sistema necessita de informações de **Feriados** (dias úteis) e de valores de **Índices Financeiros**.

Toda a movimentação dessa operação precisa ser **contabilizada** para atender às necessidades de controle interno e externo do Banco**.** E o **controle gerencial** das operações de

crédito deve ser realizado para tomadas de decisão de *marketing* e, principalmente, para o controle de Riscos do Banco.

No final da vigência da operação, ou mesmo nos vencimentos de suas parcelas, pode ocorrer de o cliente tomador da operação não conseguir cumprir suas obrigações. Nesse caso, a operação é enviada ao sistema de **Créditos Vencidos**.

## 4.2 – Definição da Arquitetura da Solução

A arquitetura a ser definida para esse exemplo de aplicação do roteiro pretende permanecer em um nível de abstração tal que possa ser implantado em diferentes plataformas e, principalmente, integrando-as.

A solução a ser dada prioriza os seguintes atributos de qualidade:

- Funcionalidade;
- Usabilidade;
- Facilidade de manutenção.

As funcionalidades devem ser completas no atendimento dos processos envolvidos na Concessão e Controle de uma operação de crédito, porém, na medida do possível, serem simples e padronizadas para os vários produtos atendidos. Quanto maior as especializações, mais complexas vão se tornando as soluções. A padronização torna a solução mais robusta e simples. Como essa solução é utilizada por diversas áreas gestoras de produtos, pelas agências e, em alguns casos, diretamente pelo cliente, a sua usabilidade deve ser simples e mais uma vez padronizada para facilitar sua utilização. A característica principal que a solução deve apresentar é a de fácil manutenção, principalmente para o Ambiente, pois como ele atende a várias aplicações, é necessário agilidade e segurança em suas manutenções, pois qualquer demora, problema ou falha nelas pode impactar várias aplicações e, conseqüentemente, vários produtos.

O presente trabalho enfoca a visão de processo do negócio e não uma visão de tecnologia e de solução física, uma vez que podem ser definidas diversas plataformas para a implementação. Com esse escopo, são detalhadas a visão lógica e a de desenvolvimento propostas por KRUCHTEN (1995). A visão física e de processo não são detalhadas no contexto desse exemplo de aplicação do roteiro.

4.2.1 – Visão Lógica da Arquitetura

A visão lógica serve para suportar os requisitos funcionais do projeto. Para atender aos Casos de Uso levantados, foi definido o seguinte Diagrama de Classes:

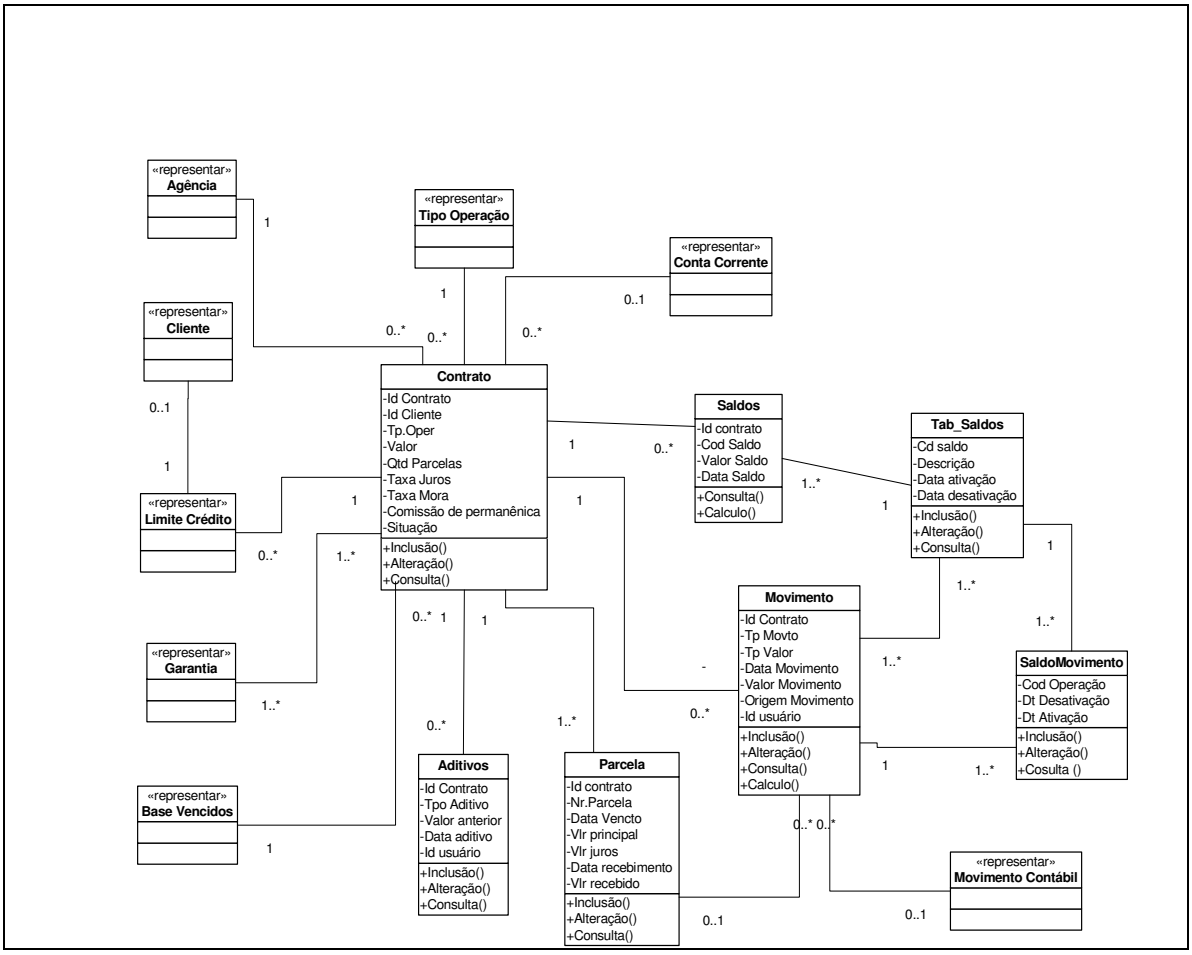



Figura 13 – Diagrama de classes – Visão funcional.

As classes definidas para o contexto da solução funcional são:

- **Contrato**: nele são registradas todas as informações para a identificação da operação de crédito, assim como as condições acordadas entre o cliente e o banco;
- **Parcela**: registro de todas as informações das parcelas geradas pela operação de crédito e as necessárias para a quitação da operação;

- **Movimento:** registro todas as movimentações que a operação sofre durante sua vigência;
- **Tabela de Saldos**: define os tipos de saldos que um determinado tipo de operação pode ter;
- **SaldoMovimento**: Classe associação que representa a relação de todos os movimentos que compõem o saldo e sua operação (soma ou subtração);
- **Saldos**: contém os valores dos saldos da operação;
- **Aditivos**: registra as mudanças que eventualmente o contrato original venha a sofrer durante sua vigência.

As classes que estão identificadas como <representar> pertencem a outros sistemas com os quais o Ambiente se relaciona, como foi apresentado no Diagrama de Contexto e só foram representadas na figura para melhor entendimento da solução.

Existe também um Diagrama de Classes que representa a visão de controle dos aplicativos derivados do Ambiente, conforme apresentado a seguir:

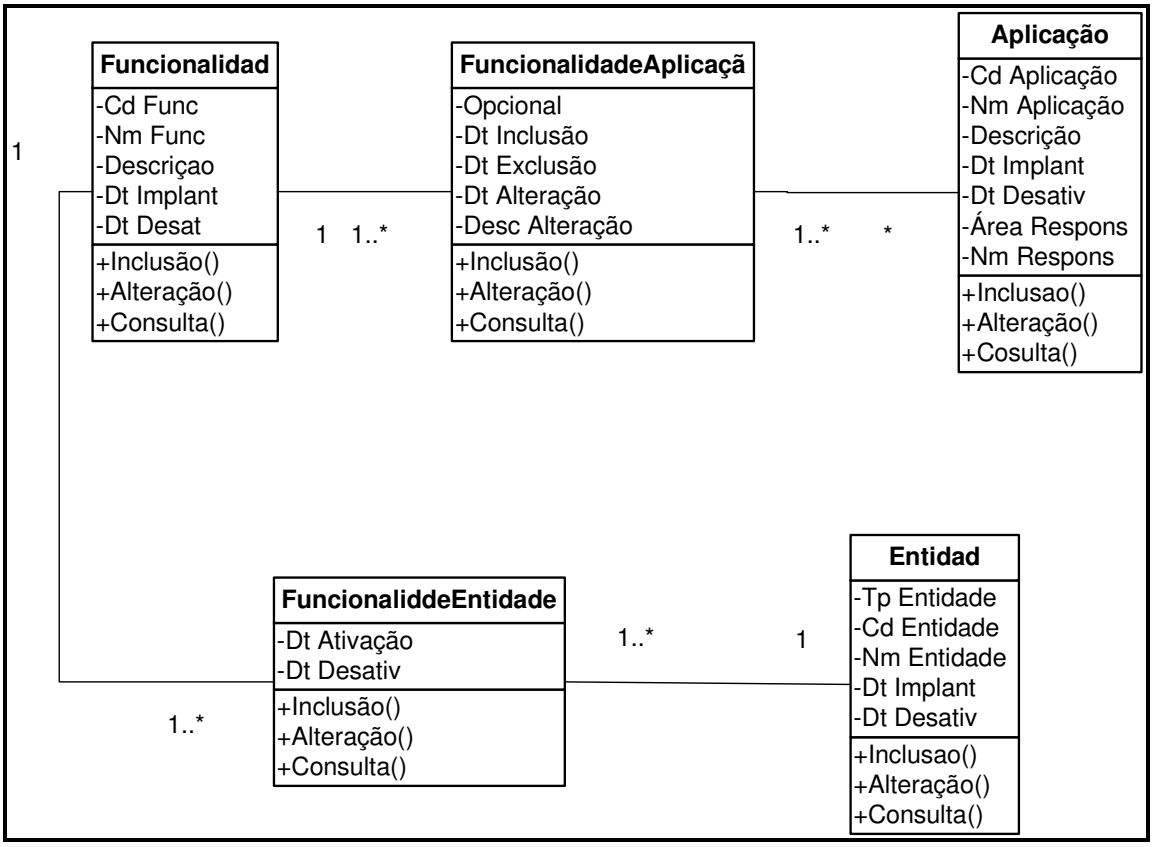



Figura 14 – Diagrama de classes – Visão controle do Ambiente.

As classes definidas para esse contexto são as seguintes:

- **Funcionalidade:** contém todas as funcionalidades que o Ambiente disponibiliza para a solução;
- **Aplicação:** representa as aplicações derivadas do Ambiente;
- **FuncionalidadeAplicação**: Classe associação que representa a relação de todas as Funcionalidades que uma Aplicação apresenta;
- **Entidade**: representa todas as entidades existentes no Ambiente. Entidades podem ser tabelas, programas, serviços, componentes, entre outros;
- **FuncionalidadeEntidade**: Classe associação que representa todas as entidades que uma funcionalidade utiliza.

Com base nessas classes, a equipe gestora do Ambiente o gerencia e gera as versões de cada Aplicação derivada desse Ambiente.

Uma outra visão interessante de ser apresentada é a visão lógica que demonstra em um nível alto de abstração as principais funcionalidades que a solução contempla. Essa visão serve para apresentar o relacionamento e dependência entre elas e deixar delimitado o escopo do projeto. Caso uma nova função não se encaixe em nenhuma funcionalidade existente, precisa ser mais bem avaliada, pois não faz parte do escopo da solução e uma análise de impacto da mudança de escopo do projeto se faz necessária. Para o exemplo em estudo, uma visão das funcionalidades é apresentada abaixo:

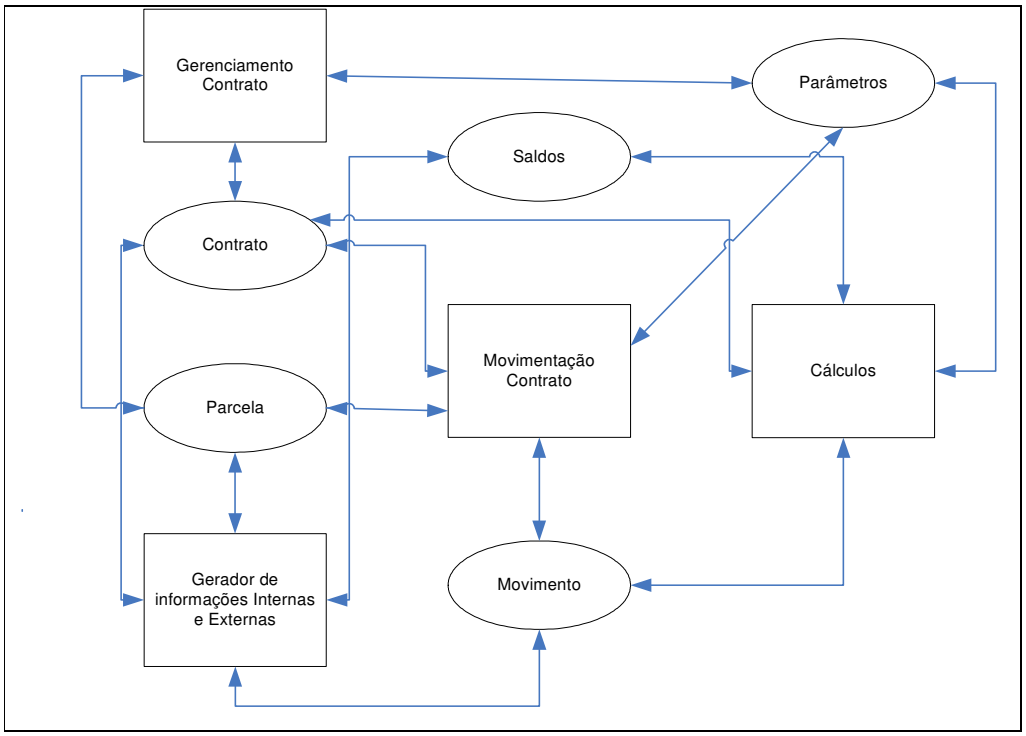



Figura 15 – Visão lógica das funcionalidades.

A visão lógica apresentada na figura 15 mostra os principais grupos de funções que compõem o Ambiente e como eles se relacionam.

A função de **Gerenciamento de Contrato** contempla as atividades de geração de contrato nas quais são definidas as informações da operação como valor, taxa de juros, vencimento e a quantidade de parcelas a serem pagas; toda a manutenção de informações que o contrato venha a ter durante sua vigência e também o seu encerramento. Ao ser gerado o

contrato, são geradas também suas parcelas, definindo o vencimento e o valor a ser pago em cada uma delas. Uma vez gerado o contrato, esse passa a receber M**ovimentações** que podem ser de liberações de valores ao tomador do empréstimo, amortizações do saldo devedor e valores gerados pelos **Cálculos** de juros, encargos e outros movimentos existentes. Com base nos movimentos, é **Calculado** o saldo do contrato conforme sua natureza de débito ou crédito. Os produtos de crédito são controlados pelo Banco Central e pelo próprio banco, por envolverem alto risco para o mercado financeiro e para o banco, por isso muitas informações do sistema são fornecidas para outros sistemas, ou são obtidas de outros sistemas via fucionalidades e **Interfaces Externas e Externas**.

4.2.2 – Visão de Desenvolvimento da Arquitetura

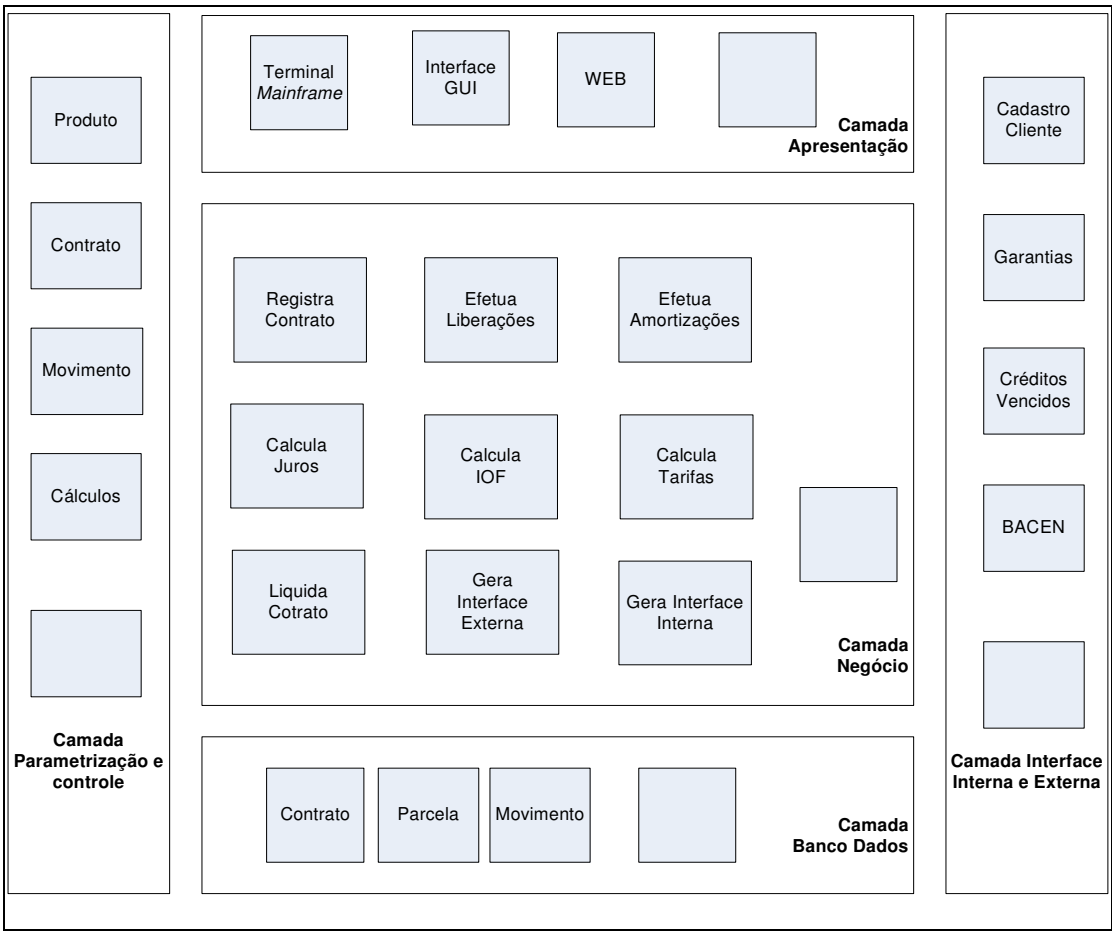



Figura 16 – Visão de desenvolvimento da arquitetura.

A visão de desenvolvimento apresenta a organização dos módulos do sistema em camadas. Essa visão permite que o sistema seja construído com as ferramentas e recursos mais adequados para cada camada e de fácil integração entre eles, o que ajuda a criar serviços e componentes com funçõs simples e específicas. A solução do exemplo apresentado está dividida nas seguintes camadas:

- *Camada de negócio*: contém todos os componentes e serviços que têm como função tratar do negócio de produtos de crédito. É o núcleo funcional do Ambiente;
- Camada de Parametrização e controle: contém as funcionalidades para que tanto o Ambiente como os produtos que são operacionalizados pelo Ambiente sejam parametrizados e controlados;
- *Camada de Interface Externa e Interna:* contém todas as funcionalidades necessárias para trocar informações com outros sistemas. Os componentes dessa camada fazem uma tradução dos padrões internos ao Ambiente para os padrões dos sistemas com os quais faz a interface;
- *Camada de Apresentação:* contém os componentes responsáveis por traduzir as informações enviadas e recebidas dos componentes da camada de negócios e por interagir com os usuários, conforme a ferramenta de apresentação definida.
- *Camada de Banco de Dados:* contém as funcionalidades relacionadas a acesso e atualizações das tabelas de banco de dados.

## 4.3 – Definição do Ambiente

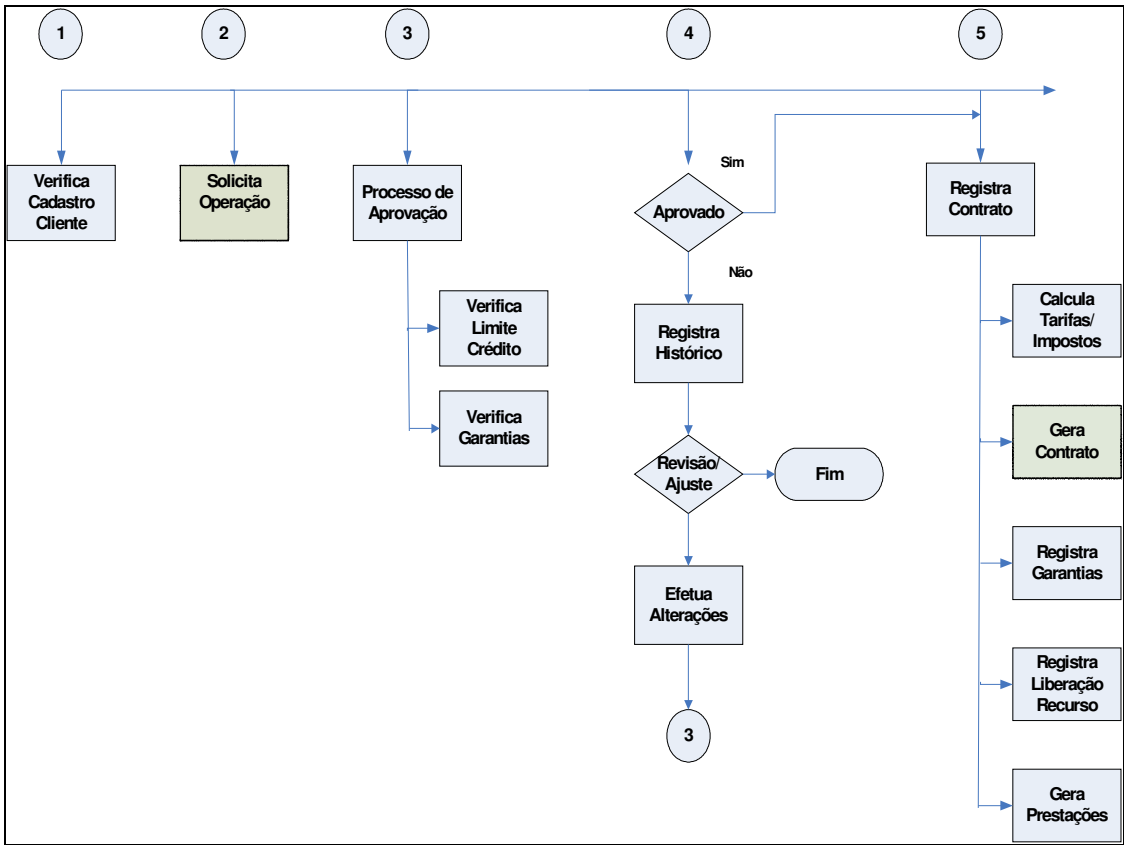
O próximo passo do roteiro é apresentar e detalhar como é construído o Ambiente, com base nas funcionalidades definidas para o domínio do problema e na arquitetura de solução proposta.

A maior parte do tempo dessa fase é utilizada para definir as funcionalidades fixas que compõem o esqueleto do Ambiente e as funcionalidades que são variáveis e adaptáveis a cada aplicação desenvolvida a partir do Ambiente. Essa definição vai apresentar o nível de

flexibilidade e adaptabilidade que o Ambiente vai ter para cada aplicação gerada e o grau de complexidade do Ambiente. Esse é um ponto vital para o sucesso da solução.

O Ambiente criado tem como escopo as funcionalidades da operacionalização de uma operação de crédito, desde sua solicitação até seu término, em normalidade com um movimento de liquidação do contrato ou, em caso de crédito vencido, na sua passagem para o sistema de Controle de Vencidos.

As Figuras 17 e 18 contêm o modelo do projeto do Ambiente a ser construído, apresentando as funcionalidades fixas em fundo azul, e as variáveis, em fundo verde.




Figua 17 - Visão geral do Ambiente – parte 1.

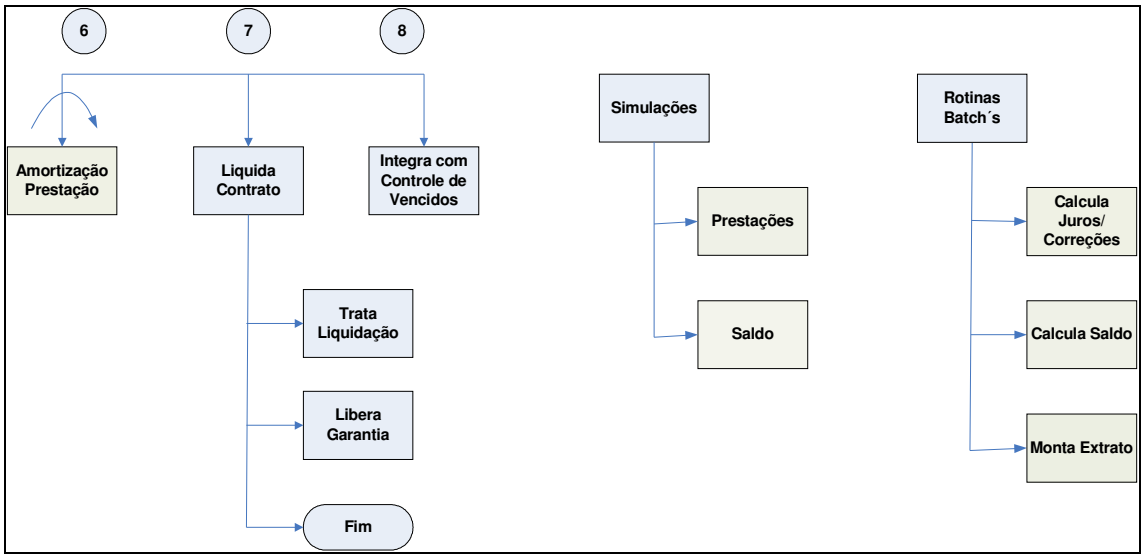



Figura 18 - Visão geral do Ambiente – parte 2.

Essas são as funcionalidades definidas para o Ambiente e que estão prontas para serem chamadas por uma camada de apresentação, a qual pode ser desde uma entrada na rede de agências, diretamente pelo cliente ou por um ponto de venda associado. Essa interface pode ser via emulação de terminal *mainframe,* por uma interface *WEB* ou Windows. Como essas funcionalidades são atendidas por serviços, eles podem ser chamados por uma carga em lote (*batch*) ou por transações *on-line*. Essa flexibilidade é facilitada pela utilização de *middlewares,* os quais viabilizam a comunicação entre os mais diversos ambientes e plataformas.

Um exemplo é o serviço de cálculos de juros que pode ser utilizado pela rotina diária de cálculo de juros, que é executada de forma *batch*, e também pela rotina de simulação de saldos, que é executada de forma *on-line*.

### 4.4 – Elaboração dos Componentes

Depois de concluídas a definição e a especificação do Ambiente e definidos os componentes que são necessários para o desenvolvimento da solução, inicia-se a fase de pesquisa/procura de componentes já existentes, de adaptação dos componentes às necessidades possíveis ou de elaboração de novos.

A solução completa dessa aplicação prática do roteiro prevê a criação de algumas dezenas de componentes. Para efeito de detalhamento, foram escolhidos os componentes relacionados na tabela a seguir que atendem às funcionalidades dos principais Casos de Uso do Ambiente.

Tabela 8 – Relação de alguns componentes.

| **Entidade** | **Nome Componente** | **Modalidade Componente** | **Descrição** |
|---|---|---|---|
| Contrato | Ctr_Modn | Consulta | Fornece informações da estrutura do Modelo de contrato para um determinado produto. |
| Contrato | Ctr_Inf_Modn | Consulta | Fornece informações de Contratos de modelo **n**. |
| Contrato | Ctr_Valid_Comum | Validação | Valida dados comuns do contrato. |
| Contrato | Ctr_Valid_Modn | Validação | Valida dados não comuns do modelo **n** do contrato. |
| Contrato | Ctr_Atu | Atualização | Atualiza registro da tabela contrato. Inclui, se não existe, e altera, se existe. Com opção de *log*. |
| Movimento | Mov_Mod | Consulta | Fornece informações da estrutura do Tipo de movimento. |

| | | | |
|---|---|---|---|
| Movimento | Mov_Inf | Consulta | Fornece informações de um ou mais movimentos de um contrato. |
| Movimento | Mov_Valid | Validação | Valida dados de um movimento. |
| Movimento | Mov_Atu | Atualização | Atualiza registro de tabela de movimento. Inclui, se não existe, e altera, se existe. Com opção de *log*. |
| Movimento | Calc_IOF | Cálculo | Calcula o valor do IOF. |
| Movimento | Calc_Juros_Pré | Cálculo | Calcula Juros dos contratos pré-fixados. |
| Movimento | Calc_Juros_Pós | Cálculo | Calcula Juros dos contratos pós-fixados. |
| Saldo | Calc_saldo | Cálculo | Calcula Saldo do contrato. |
| Cliente | Cli_Inf | Interface | Solicita dados de clientes. |
| Garantia | Gar_Reg | Interface | Solicita o registro das garantias de um contrato. |

Os componentes devem ser desenvolvidos seguindo os padrões definidos do projeto para que eles possam ser facilmente chamados pelo Ambiente, o que garante um melhor sincronismo na criação desse Ambiente.

É importante que seja utilizado o máximo possível de padrões. Os padrões básicos a serem utilizados são de nomenclatura de campos, atributos artefatos gerados, da estrutura de interface e, se possível, de estutura de programas. A utilização de padrões facilita o entendimento e a utilização do Ambiente.

### 4.5 – Elaboração do Ambiente

Como já mencionado anteriormente, o objetivo desse trabalho é apresentar um roteiro de desenvolvimento, sendo que não faz parte de seu escopo a construção de nenhum componente. É importante enfatizar que, até essa etapa do roteiro, a plataforma onde será construído o Ambiente não influenciou a sua modelagem. A partir dessa fase do roteiro, deve ser definida a plataforma em que é desenvolvida e desenhada a arquitetura física, sem, portanto, alterar o núcleo da arquitetura definida.

De forma a manter esse Ambiente íntegro, com seus propósitos de qualidade quanto à reusabilidade/usabilidade e facilidade de manutenção, um gerenciamento constante pelo zelo dessas qualidades se faz necessário. Esse gerenciamento visa a não permitir que duplicidades de funcionalidades aconteçam ou que, caso aconteçam, que sejam de forma consciente e justificável. Também é importante que funcionalidades sejam desenvolvidas na camada correta para que seu escopo seja claro, objetivo e simples. O desenvolvimento de funcionalidades em local não adequado aumenta a probabilidade de um componente perder seu objetivo e, principalmente, sua simplicidade, visto que, para fazer o que precisa no lugar não adequado, muitas vezes acaba ganhando funções de correção/adequações que, se estivessem em local apropriado, não seriam necessárias.

A escolha da tecnologia em que o Ambiente é construído deve ser criteriosa, de forma a permitir a interoperabilidade de plataformas, desempenho e produtividade de desenvolvimento. A tecnologia adotada tem que ser uma facilitadora para desenvolver essas qualidades e não um fator limitador.

É preciso ter claro que, para o Ambiente ganhar uma estabilidade adequada, ele precisa ter passado pelo processo de desenvolvimento de algumas aplicações. No decorrer dessa fase de gerações das aplicações iniciais, ocorre naturalmente um refinamento do Ambiente,

É necessário manter uma equipe permanente de desenvolvimento com recursos humanos altamente capacitados, treinados e conhecedores da solução do Ambiente. O desenvolvimento das aplicações pode e deve ser feito por equipe distinta da equipe do Ambiente, uma vez que o perfil e as preocupações exigidas para cada uma delas são bem diferentes. Enquanto a equipe do Ambiente deve estar focada em criar serviços reusáveis e genéricos, a equipe das aplicações deve estar focada em especializar a solução, seguindo os padrões definidos pelo Ambiente.

## 4.6 – Desenvolvimento da Aplicação

Uma vez desenvolvido o Ambiente, é importante validar e testar sua solução e sua arquitetura com a implementação de uma ou mais de uma aplicação baseada no Ambiente.

Para que a decisão de desenvolver uma aplicação a partir do Ambiente seja tomada, é importante que o domínio da aplicação seja conhecido e tenha passado por um *check-list* pré-definido de itens do Ambiente que indiquem a viabilidade desse desenvolvimento a partir desse Ambiente.

É esperado que, nas primeiras aplicações desenvolvidas a partir do Ambiente, esse venha a ter necessidade de ajustes. Porém, por volta do terceiro aplicativo, uma estabilidade do Ambiente precisa ser notada, para que os objetivos de desenvolvimento rápido e com qualidade sejam alcançados.

Uma vez tomada a decisão de desenvolver a aplicação a partir do Ambiente, é necessário levantar e definir as partes variáveis que devem ser desenvolvidas e que são ligadas a ele.

Para essa aplicação prática do roteiro, é apresentada a solução para duas aplicações utilizando o Ambiente criado, bem como apresentando as implementações necessárias para atender às necessidades dessas aplicações.

As duas aplicações são para atender aos produtos de Cheque Especial e de Crédito Direto ao Consumidor – CDC. Ambos foram escolhidos por apresentarem características diferentes na sua operacionalização e por serem modalidade de produtos de crédito simples e de conhecimento amplo das pessoas. A maioria das pessoas tem um cheque especial ou já fez um CDC para adquirir algum bem.

Podem ser relacionadas as seguintes principais diferenças entre esses produtos:

Tabela 9 – Diferenças entre os produtos CDC e Cheque Especial.

| **Característica** | **Cheque Especial** | **CDC** |
|---|---|---|
| Liberação do montante | Liberações parciais dentro do limite aprovado. | Liberação única. |
| Controle de Saldo | Informado pelo sistema de conta-corrente. | Calculado pelo sistema. |

| | | |
|---|---|---|
| Garantia | Nota Promissória | É o próprio bem envolvido na operação |
| Número de parcelas | Não tem. | Definido na formalização do contrato. |
| Liquidação da operação | Não tem. | No pagamento da última parcela. |

Abaixo são apresentadas as aplicações necessárias para operacionalizar os produtos propostos:

**a) Desenvolvimento da Aplicação 1**

A aplicação 1 atende ao produto de CDC. Por parametrização, todas as funcionalidades do Ambiente são utilizadas e os módulos variáveis são desenvolvidos conforme as figuras a seguir:

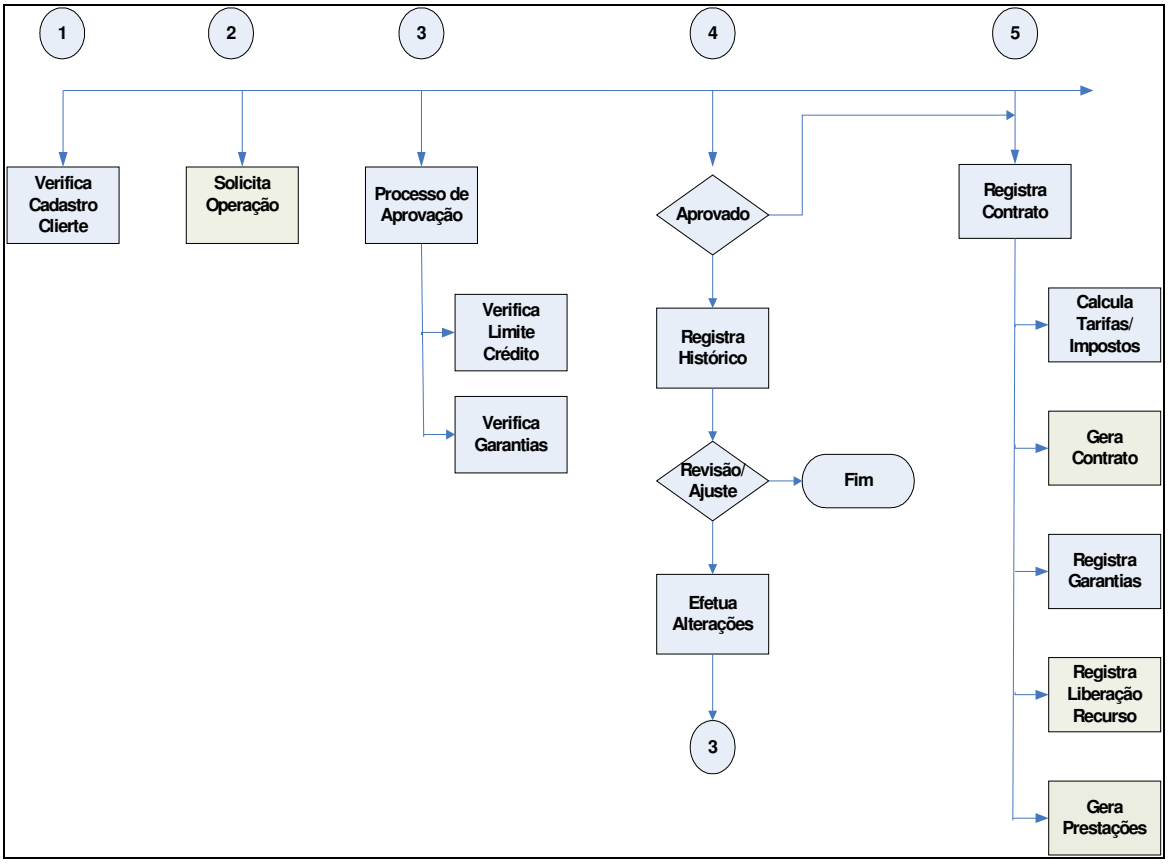



Figura 19 – Visão da aplicação para produto CDC – parte 1.

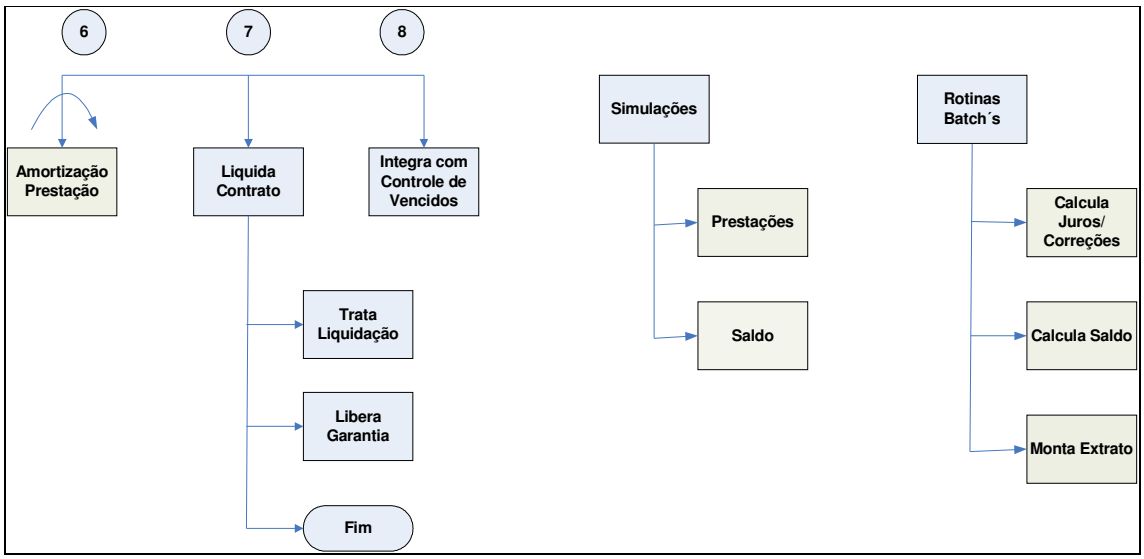



Figura 20 – Visão da aplicação para produto CDC – parte 2.

As funcionalidades especializadas desse produto são as seguintes:

- **Solicita Operação**: Para registrar uma solicitação de operação de CDC, é necessária a informação dos seguintes campos: Identificação do cliente (código cliente ou CPF/CNPJ), tipo de operação, valor solicitado, número de parcelas e taxa de juros.
- **Gera Contrato:** O registro do contrato de CDC solicita o registro dos seguintes campos: os fornecidos na solicitação da operação, tipo de cobrança, tipo pagamento e IOF.
- **Registra Liberação Recurso:** O valor de empréstimo contratado menos os valores dos encargos, tarifas, comissões e IOF são liberados para o cliente via crédito em conta-corrente ou da forma como foi negociado.
- **Gera Prestações:** As prestações devem ser geradas conforme quantidade e condições fechadas na contratação do produto.
- **Amortização Prestação:** É gerado um movimento de amortização cada vez que o cliente efetuar o pagamento de uma prestação.
- **Cálculo dos Juros/Correções:** Os juros podem ser pré ou pós-fixados, conforme estabelecido na formalização do contrato.
- **Cálculo do saldo:** O saldo calculado pela posição do saldo anterior e somando e subtraindo os movimentos do contrato, conforme suas características sejam de movimento de débito ou de crédito. Os movimentos e suas características são cadastrados para cada produto em uma tabela que é acessada no momento do cálculo.
- **Monta Extrato:** É demonstrada a vida do contrato, com todas suas as características e movimentações sofridas e com o saldo atualizado.
- **Simulação de Prestações e Saldo:** Aplica-se a rotina de cálculo de juros e de correções sobre os valores e sobre a quantidade de parcelas informadas na simulação.

**b) Desenvolvimento da Aplicação 2**

A aplicação 2 atende ao produto de Cheque Especial. Por parametrização, nem todas as funcionalidades do Ambiente são utilizadas e os módulos variáveis são desenvolvidos conforme as figuras a seguir:

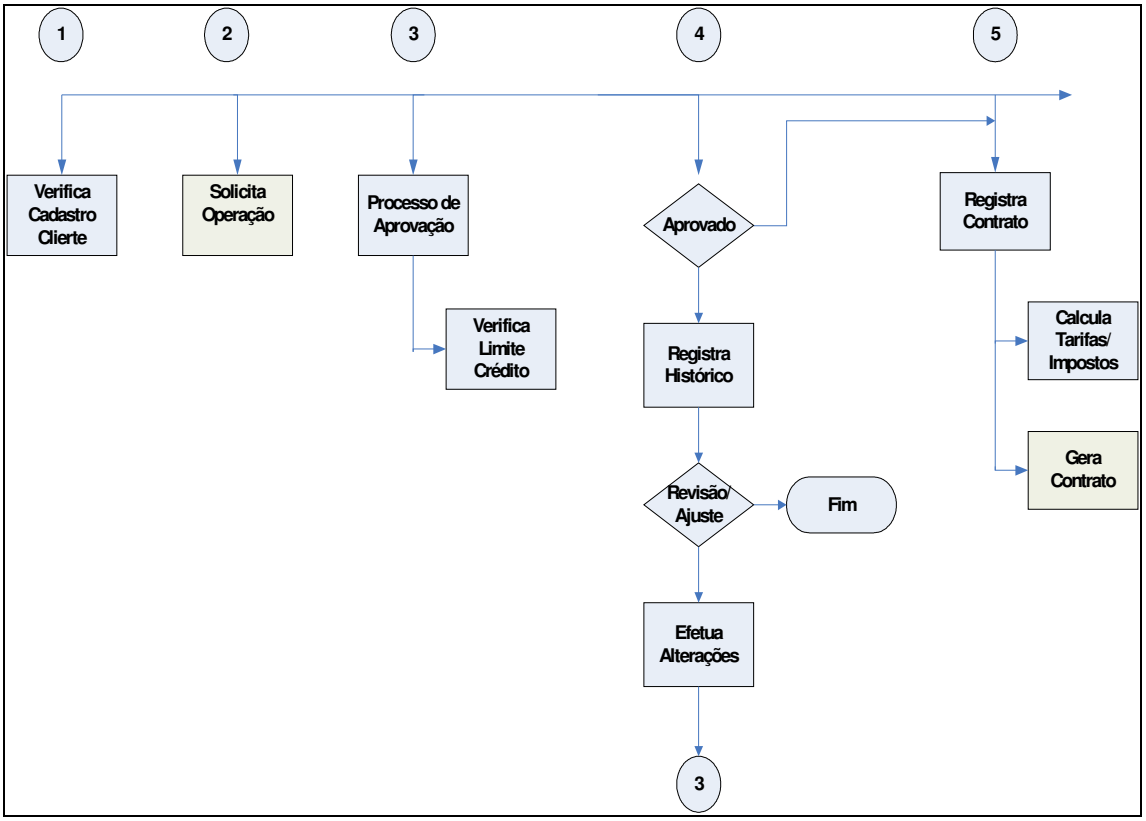



Figura 21 – Visão da aplicação para o produto Cheque Especial – parte 1.

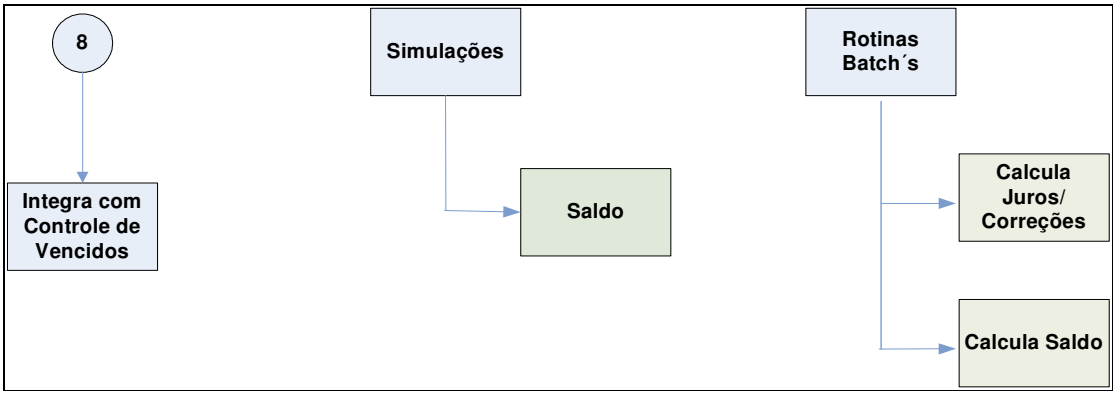



Figura 22 – Visão da aplicação para o produto Cheque Especial – parte 2.

As funcionalidades desse produto são as seguintes:

- **Solicita Operação**: Para registrar uma solicitação de operação de Cheque Especial, é necessária a informação dos seguintes campos: número da conta corrente, o valor do limite de cheque especial aprovado e a data de vencimento desse limite.
- **Gera Contrato:** O registro do contrato de Cheque Especial solicita o registro dos seguintes campos: o número da conta corrente, o limite de crédito aprovado e a data de vencimento do limite.
- **Calcula Juros e Correções:** É sempre pré-fixado.
- **Calcula Saldo:** Informado pelo sistema de Conta-corrente.
- **Simulação de Saldo:** Aplica-se a rotina de cálculo de juros e correções sobre valores informados na simulação. Não existe parcela para Cheque Especial.

## 4.7 – Comentários sobre a Aplicação do Roteiro

A aplicação prática do roteiro mostrou que o mesmo ficou simples de ser aplicado, consolidando técnicas e artefatos já existentes e utilizados pelos desenvolvedores de soluções de *software*.

Uma atenção especial deve ser dada à etapa inicial da Definição do Domínio do Problema, pois a Análise de Domínio é o ponto de partida para a construção de um projeto de sucesso e essa é uma atividade que, muitas vezes, é negligenciada pela equipe do projeto, principalmente se ela tiver um perfil muito técnico. Os conceitos e técnicas de Análise de Domínio, se seguidos, produzem um conhecimento sobre o assunto em questão que é fundamental para as definições das próximas etapas.

A definição da arquitetura da solução deve considerar os requisitos funcionais e os não funcionais do projeto. Ela vai estabelecer a estrutura e a flexibilidade para a evolução do projeto.

A definição dos componentes deve ser criteriosa e, até certo ponto, rigorosa, no sentido que deve seguir e ditar regras para sua utilização, o que torna a utilização do Ambiente para a geração de novas aplicações claro e estável.

A mistura de técnicas e modelos como UML, análise estrutrada ou alguma própria da empresa pode e deve ocorrer sem constrangimentos, uma vez que o importante é conseguir documentar e tornar claro os resultados de cada fase de desenvolvimento do roteiro.

# Capítulo 5 – Considerações Finais

## 5.1 – Resultados Obtidos

Esse trabalho apresenta um roteiro para a criação de um Ambiente baseado em componentes para uma família de produtos.

O objetivo do estudo é capturar conceitos e conhecimentos existentes de desenvolvimento baseado em componentes e reuso e aplicá-los, de forma direcionada e objetiva, para uma família de produtos. Esse direcionamento se apresenta como conseqüência da tentativa de incentivar a utilização desses conceitos de uma forma mais especializada. Na área bancária, a componentização e reuso é geralmente aplicada na camada que abrange os sistemas corporativos, em soluções voltadas para cadastro de Clientes e algumas tabelas como as de Agências e de feriados.

Pesa ainda o fato que, na área bancária, para bancos de médio e grande porte, ainda se utilizam soluções voltadas para *mainframe*, nas quais a aplicação de componentização é, infelizmente, pouco utilizada. Essa baixa utilização ocorre por alguns fatores:

- Sistemas antigos. Muitos foram desenvolvidos há décadas quando muitos conceitos e técnicas de reuso e de componentização não existiam;
- Por ser um ambiente antigo, os profissionais que detêm conhecimento e que o desenvolvem nesse ambiente, em geral, são profissionais com formação também antiga e que nem sempre foram se atualizando com as novas técnicas e conceitos;
- Ferramentas e linguagens para desenvolvimento em *mainframe* são comumente menos flexíveis e com menos recursos que as de baixa plataforma;
- A bibliografia existente sobre o assunto é basicamente encontrada para um ambiente com solução voltada a Orientação a Objeto, sendo que, no *mainframe,* essa solução é pouco utilizada, visto ser preferida a solução de Análise Estruturada.

Como o objetivo de desenvolver *softwares* com mais qualidade e em menor tempo é comum para qualquer plataforma e em qualquer área de negócios, esse trabalho visa a aplicar

as técnicas de componentização e reuso independentemente da tecnologia utilizada para o desenvolvimento da aplicação.

Esse é o diferencial do trabalho proposto, aplicar as técnicas e conceitos de reuso e componentização focando o negócio e não a tecnologia, de forma que possam ser utilizadas em qualquer solução que comporte a aplicação. Tal postura se deve ao fato que, apesar de uma tecnologia adequada facilitar sua utilização, o fator cultural de desenvolvimento ainda é o maior impeditivo para o seu uso.

## 5.2 – Contribuições da Pesquisa

O roteiro proposto nesse trabalho visa a ser bastante simples, utilizando técnicas já conhecidas. Um diferencial deste trabalho está no destaque dado à visão de negócio, na qual tudo se inicia com uma Análise de Domínios e caminha pelas fases tradicionais de desenvolvimento de sistemas, focando sempre o negócio e o processo em primeiro lugar e, somente no momento adequado, a parte de tecnologia.

Essa mudança de foco pretende viabilizar a utilização do roteiro em processo de desenvolvimento para qualquer plataforma de solução e não somente para as já tradicionais soluções Orientadas a Objeto para arquiteturas *Client Server* ou *Web.*

O roteiro apresentado é genérico e pode sofrer adaptações visando melhorias à medida que for aplicado em família de produtos específicos.

## 5.3 – Próximos Passos

O presente trabalho apresenta um roteiro para a criação de um Ambiente baseado em componentes para uma família de produtos. Esse roteiro tem condições de ser mais detalhado e tornar-se um processo e, mais ainda, uma metodologia, ao ser completado com o envolvimento de assuntos como estrutura organizacional e funcional necessária, ferramentas associadas, ciclo de vida do Ambiente, dos componentes e das aplicações criadas a partir do Ambiente.

Outros aspectos desta dissertação que podem ser explorados em trabalhos futuros, sendo abordados com mais detalhes, são:

- Definição de processos para implementação de um processo de desenvolvimento de *software* voltado para reuso;

- Definição de processos para a implementação de um processo de desenvolvimento de *software* voltado para família de produtos;

Esses são pontos críticos e que envolvem mais aspectos de processos e de estrutura organizacional do que aspectos técnicos. Somente a utilização de ferramentas e conceitos isolados, sem um processo suportado por uma estrutura organizacional, acaba não trazendo os resultados esperados de qualidade e rapidez no desenvolvimento de *software*.

## Referências Bibliográficas


- BASS, L.;CLEMENTS, P.;KAZMAN, R. **Software Architecture in practice**. 2.e. Addison-Wesley, 2003.
- CODENIE, W.;STEYAERT, P.; VERCAMMEN, A. **Evolving Custom-Made Applications into Domain-Specific Frameworks**. Disponível em: <http:/citeseer.ist.psu.edu/codenie97evolving.html>. Acesso em: 21.02.2006.
- COMER, E.R. **Domain Analysis: A system approach to software reuse**. Digital Avionics Systems Conference. Proceedings IEEE/AIAA/NASA 9th**.** 15-18 Pct.1990.Page(s): 224-229.
- CRNKOVIC, I.; LARSSON, M. **A Case Study: Demands on Component-based Development**. Proceedings of the 22nd Internacional Conference on Software Engineering. ACM Press. June, 2000.
- D´SOUZA, D.F.; WILLS, A.C. **Objects, components, and frameworks with UML : the Catalysis**. Addison-Wesley, 1998.
- ESTUBLIER, J.; VEGA, G. **Reuse and Variability in Large Software Applications.** ACM SIGSOFT Software Engineering Note, Proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium of Foundations of Software engineering ESEC/FSE-13, vol.30 issue 5. ACM Press. September, 2005.
- GARLAN, D. **Software Architecture: a Roadmap**. ACM SigSoft Enginnering Notes. May, 2000.
- ISO/IEC 9126-1, **Software Engeneering - Product Quality – Part 1: Quality Model**, 2001.
- KRUCHTEN, P. **Architectural Blueprints - The '4+1' View Model of software Architecture**. Rational Software Corp. 1995.
- KRUEGER, W. C. **Software Reuse.** ACM Computing Surveys, vol.24 issue 2. 1992
- LEVESON, N.G.; WEISS, K.A. **Making Embedded Software Reuse Practical and Safe**. ACM SIGSOFT Software Engineering Notes, Proceedings of the 12th ACM SigSoft twelfth international symposium on Foundations of software engineering. SIGSOFT '04/FSE-12, Volume 29 Issue 6. October, 2004.

- LIN, M.; YONGSEN, X. **An adaptive Dependability Model Of Component-Based Software**. ACM SigSoft Enginnering Notes Vol 28 Issue 2. March, 2003.
- MENDES, A. **Arquitetura de *Software* : Desenvolvimento orientado para arquitetura**. Rio de Janeiro: Campus, 2002.
- NIEMELÃ, E.; IHME, T. **Product Line Software Engineering of Embedded Systems**. In: Proceedings of the 2001 Symposium on Software Reusability: putting software reuse in context SS2'01, volume 26, issue 3. ACM Sigsoft. May, 2001.
- PIETRO-DIAZ, R. **Domain Analysis – An Introduction**. ACM SigSoft Enginnering Notes. Apr, 1990.
- PRESSMAN, R.S. – **Engenharia de *Software***. 5.d. Rio de Janeiro: McGraw Hill, 2002.
- ROSSI, A.C. **Representação do Componente de *Software* na FARCSOFT:Ferramenta de apoio à reutilização de Componentes de *Software***. 2004. 237 f. Dissertação(Mestrado em Engenharia), Escola Politécnica da Universidade de São Paulo, São Paulo – SP, 2004.
- SOMMERVILLE, I. **Software Engineering**. 6.e. Addison-Wesley Publishers, 2001.
- VAROTO, A.C. **Visões em arquitetura de *software***. 2002. 99 f. Dissertação(Mestrado em Ciência da Computação), Instituto de Matemática de Estatística, Universidade de São Paulo, São Paulo - SP, 2002.
- WILLIAMS, J.D. **Component Framework worth the Struggle**. Feb, 2001. Disponível em: <www.adtmag.com/print.asp?id=2676>. Acesso em: 30.06.2004.