WAGNER LEAL DOS SANTOS

UM PROCESSO DE MIGRAÇÃO DE SISTEMA LEGADO FUNCIONAL PARA ORIENTADO A OBJETOS DIRECIONADO POR INDICADORES DE QUALIDADE

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para a obtenção do título de Mestre em Engenharia

SÃO PAULO
2007

WAGNER LEAL DOS SANTOS

UM PROCESSO DE MIGRAÇÃO DE SISTEMA LEGADO FUNCIONAL PARA ORIENTADO A OBJETOS DIRECIONADO POR INDICADORES DE QUALIDADE

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para a obtenção do título de Mestre em Engenharia

Área de Concentração:
Sistemas Digitais

Orientador:
Prof. Dr. Reginaldo Arakaki

SÃO PAULO
2007

FICHA CATALOGRÁFICA

Santos, Wagner Leal dos
Um processo de migração de sistema legado funcional para orientado a objetos direcionado por indicadores de qualidade / W.L. dos Santos. -- ed.rev. -- São Paulo, 2007.
p.

Dissertação (Mestrado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais.

1.Migração de sistemas legados 2.Arquitetura de software 3.Qualidade de software 4.Reengenharia I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Computação e Sistemas Digitais II.t.

Dedico este trabalho aos meus pais, José Raymundo dos Santos e Sonia Maria Leal dos Santos, pela dádiva da existência e pelo infinito apoio em todos os momentos de minha vida.

AGRADECIMENTOS

Agradeço a Deus pela vida, saúde e por propiciar oportunidades de crescimento em todas as minhas realizações.

Às minhas irmãs pela garra, perseverança e dignidade nos momentos de dificuldade.

Ao meu orientador, Reginaldo Arakaki, pelo apoio, paciência e também por transmitir sua experiência e por me fazer perceber a importância da arquitetura de software.

Ao professor Jorge Risco pelo incentivo e pelos momentos de discussão que contribuíram muito para esta realização.

A todos os alunos de pós-graduação da Poli/USP que, direta ou indiretamente, colaboraram na execução deste trabalho, em especial aos amigos Dimas Ribeiro de Magalhães, André Luiz Dias Ribeiro, Renato Manzan de Andrade, Giuliano Caliari e Beatriz Borsoi pelas contribuições e indicações.

Aos meus amigos do Banco Itaú, especialmente, Cássio Henrique Campos, Beatriz Nogueira, João Roberto Salles, Ricardo Haluska, Arlindo Manoel, Eduardo Padilha, Valmir Fernandes Junior, Silvia Cobo, Sumara Vieira e Márcia Maria Chirano que me proporcionaram, pelos anos de convivência e camaradagem, muitas oportunidades de aprendizado e pelo apoio nessa empreitada.

Às minhas queridas esposa e filha Maria Lucia Novais e Letícia Novais dos Santos pelo amor, companheirismo, compreensão e constante incentivo durante toda esta etapa de nossas vidas.

RESUMO

A manutenção de sistemas legados tem se tornado uma preocupação constante das grandes empresas. O setor bancário brasileiro, por exemplo, possui milhões de linhas de código confeccionados em linguagens procedurais, essenciais para atingir os objetivos de negócios destas instituições. Muitos desses programas são considerados bem antigos, possuindo mais de 30 anos de existência e, apesar de serem extremamente úteis para estas organizações, não permitem aproveitar as vantagens das novas tecnologias, tais como o uso de interfaces gráficas, processamento distribuído, entre outros. Fazer um outro aplicativo de software a partir do início pode ser uma tarefa muito árdua e incorrer em grandes riscos para o negócio da empresa. Migrar esses sistemas aos poucos parece ser a melhor estratégia. Isso porque, a utilização dessa abordagem permite que a adaptação dos usuários ao novo sistema seja gradativa, ou seja, ocorre à medida que as funcionalidades de negócio são contempladas pelo novo sistema. Considerando a necessidade evidente que essa migração terá de ocorrer mais cedo ou mais tarde, este trabalho propõe um processo para permitir uma evolução gradual do software legado para uma plataforma mais moderna e de mais amplo uso atualmente, que possa atender melhor às novas necessidades dos negócios. Esse processo é formado por etapas de avaliação da situação atual do sistema, de transformação de arquitetura e de transformação do código funcional para o orientado a objetos. Além disso, é direcionado por indicadores de qualidade e apoiado por tipos de visão e respectivos estilos arquiteturais. O foco deste trabalho está em sistemas de instituições financeiras desenvolvidos em *Mainframe* ou AS/400, onde há grande incidência de códigos antigos orientados a procedimentos.

ABSTRACT

The maintenance of old legacies software has become a constant concern of the great companies. The Brazilian banking sector, for example, has millions of lines of code made under the functional paradigm, essential to reach the business-oriented objectives of these institutions. Many of these systems are very old, arriving to possess more than 30 years of existence. These programs, extremely useful for these organizations, do not allow the use of the advantages of the new technologies, such as the use of graphical interfaces, distributed processing and so on. The replacement of all old system for a new on may be a very arduous task and to incur into great risks for the company. Migrate these systems in small steps seems to a better strategy to deal with this problem that will have to be faced earlier or later. This work considers process to allow the gradual evolution of the legacy system to a better platform that allows the use of the advantages of the new technologies, through the use of quality indicators, the evaluation of the current system, the transformation of the architecture with focus at the change of functional paradigm to the object oriented one and of the use of view types and the corresponding architectural styles. This work is mainly delivered to financial institutions systems developed in Mainframe or AS/400 that have great incidence of these old legacies systems.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE QUADROS

CAPÍTULO 1

INTRODUÇÃO

Objetivo do Capítulo

O objetivo deste capítulo é apresentar as motivações e justificativas que levaram ao desenvolvimento deste trabalho, além da metodologia utilizada e a estruturação da dissertação.

1.1. O Problema da Manutenção dos Sistemas Legados

Atualmente existem muitos sistemas desenvolvidos sob o paradigma orientado a procedimentos. Estes programas foram desenvolvidos em linguagens como COBOL, RPG, por exemplo, e por profissionais formados (pela universidade ou pela empresa) para desenvolver programas batch e on-line sob o paradigma orientado a procedimentos.

Na medida em que estas aplicações são desenvolvidas para atender novos requisitos, elas vão se tornando cada vez mais complexas, encarecendo o custo de desenvolvimento de novas solicitações.

No caso de muitas aplicações bancárias, os softwares legados muitas vezes estão "integrados" com outros sistemas em diferentes plataformas, distribuídos em diversos hardwares dispersos ao longo das organizações.

A manutenção de software legados, desenvolvidos sob o paradigma procedural, pode, em vista dos fatos relatados nos parágrafos anteriores, dificultar a redução de custos e prazos, limitar ações de melhoria de qualidade voltadas ao atendimento de requisitos não funcionais.

1.2. Motivação

A manutenção de sistemas legados tem se tornado uma preocupação constante das grandes empresas, inclusive no Brasil. O setor bancário brasileiro, por exemplo, possui milhões de linhas de código confeccionados sob o paradigma funcional codificadas em linguagens como COBOL.

Esses programas, apesar de serem essenciais para atingir os objetivos de negócios destas instituições, não permitem aproveitar melhores vantagens oferecidas pelas novas tecnologias e metodologias voltadas ao paradigma da Orientação a Objetos, tais como o uso de novos tipos de interfaces e melhor tratamento da complexidade e da distribuição dos componentes.

Dado a rápida evolução tecnológica do hardware, onde temos o uso de interfaces mais sofisticadas (*wireless*, reconhecimento biométrico, etc*), a* sua integração com o software é cada vez mais constante. Essas novas tecnologias aliadas à crescente competitividade de mercado, às mudanças de requisitos durante o projeto e à necessidade de redução do tempo de desenvolvimento demandam um aumento de produtividade, qualidade e complexidade de software.

Conforme publicado em artigos como Battaglia et al. (1998), entre outros, softwares desenvolvidos sob a ótica da orientação a objetos (OO) permitem uma maior facilidade de manutenção, de distribuição entre usuários (pode ser acessado via *browser*) e de integração com outros aplicativos (*Web Services*). Desta forma, é razoável avaliar a viabilidade de migrar sistemas desenvolvidos sob a ótica de procedimentos para o paradigma OO.

1.3. Objetivos do Trabalho

O principal objetivo deste trabalho é propor a organização de práticas e técnicas de engenharia de s*oftware* em um processo aplicável que permita a evolução tecnológica do software com redução do tempo no desenvolvimento e melhoria da qualidade do *software.*

No processo, é descrita a utilização planejada das práticas de engenharia de *software,* tais como desenvolvimento incremental, utilização de visões e estilos arquiteturais e captura de requisitos funcionais e não funcionais, que auxiliam na definição dos indicadores de qualidade, na elaboração do modelo arquitetural do sistema atual, na proposição de um novo modelo de arquitetura e na transformação arquitetural de um sistema legado funcional para um sistema orientado a objetos.

1.4. Metodologia

Os aspectos metodológicos considerados para o desenvolvimento destes trabalhos compreendem as seguintes atividades:

1) Estudo de aspectos conceituais relacionados com a manutenção e evolução de software, qualidade de software e arquitetura de software;
2) Estudo de pesquisas relacionados a migração de software legado;
3) Proposição de processo de transformação de arquitetura de software com base nas pesquisas realizadas;
4) Aplicação do processo em um exemplo utilizando um sistema de cartões magnéticos;
5) Avaliação do processo proposto.

A primeira atividade se refere à pesquisa bibliográfica correspondente ao estudo das referências teóricas sobre os principais conceitos relacionados à Arquitetura, Manutenção, Evolução e Qualidade de Software.

A segunda atividade consiste em verificar as abordagens existentes sobre a conversão de Sistemas Funcionais em Orientados a Objetos visando à melhoria da qualidade e redução de tempo de manutenção e evolução de software.

A proposição do processo de transformação de arquitetura de sistema legado funcional para orientado a objetos, visando redução do tempo de manutenção e melhoria dos indicadores de qualidade que, compreende a elaboração do processo propriamente dito, considerando os aspectos técnicos estudados, constituem os objetivos da terceira atividade.

A quarta atividade tem como objetivo realizar a aplicação do processo proposto em projetos de software acompanhados pelo autor em sua atividade profissional.

Na última atividade, a avaliação do processo proposto, consiste em analisar os resultados coletados durante a aplicação do processo.

1.5. Organização do Trabalho

Este trabalho foi organizado em sete capítulos conforme representação esquemática da Figura 1.1.

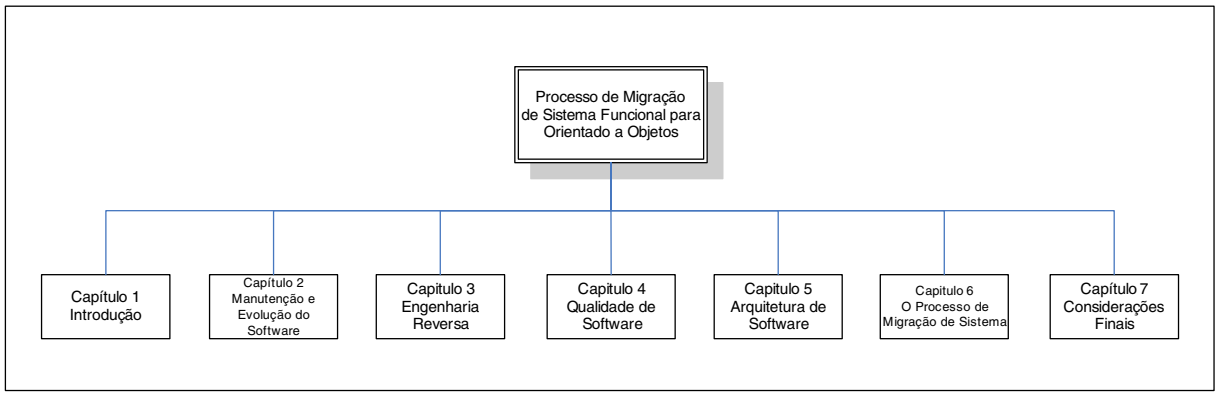

Figura 1.1 – Representação gráfica da organização do trabalho

Os motivos que levaram ao desenvolvimento deste trabalho, seus objetivos, a metodologia utilizada, a estruturação do trabalho e as convenções utilizadas são apresentados neste **Capítulo 1**.

No **Capítulo 2** são descritas as etapas envolvidas na manutenção e evolução de software e aspectos que devem ser considerados nestas atividades.

No **Capítulo 3** são descritos os aspectos conceituais da engenharia reversa e sua importância para a recuperação do modelo arquitetural do sistema.

No **Capítulo 4** é descrita a fundamentação teórica dos conceitos das técnicas e práticas relacionadas à qualidade de software.

No **Capítulo 5** o foco é a fundamentação teórica dos conceitos, das técnicas e das práticas relacionadas a arquitetura de software.

No **Capítulo 6** é apresentado o processo para transformação de arquitetura de software de sistemas legados funcionais para orientados a objetos, direcionados por indicadores de qualidade.

No **Capítulo 7** são apresentadas as considerações finais, as conclusões e as contribuições do trabalho.

1.6. Notações e Convenções Utilizadas

São apresentadas abaixo a listas de abreviaturas:

ABNT	Associação Brasileira de Normas Técnicas
DFD	Diagrama de Fluxo de dados
DER	Diagrama Entidade-Relacionamento
IDEF0	Integration Definition for Function Modeling
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IPT	Instituto de Pesquisas Tecnológicas do Estado de São Paulo
ISO	International Organization for Standardization
NBR	Norma Brasileira
OMG	Object Management Group
UML	Unified Modeling Language
USP	Universidade de São Paulo
XML	Extensible Markup Language

As tabelas e figuras que não possuem referência foram desenvolvidas pelo autor utilizando diagramas UML – *Unified Modeling Language* (OMG), representações clássicas de fluxograma, diagrama entidade-relacionamento (DER) ou IDEF0 – Integration Definition for Function Modeling.

CAPÍTULO 2

MANUTENÇÃO E EVOLUÇÃO DO SOFTWARE

Objetivo do Capítulo

O objetivo deste capítulo é descrever os aspectos envolvidos com a manutenção e a evolução do software, as etapas e os aspectos a serem considerados, e sua importância dentro do ciclo de desenvolvimento de sistemas.

Este capítulo foi desenvolvido com base no estudo dos seguintes trabalhos:

Magalhães e Arakaki (2005); Souza e Arakaki (2004); Pfleeger (2004); Presman (2002); Cânfora (1997);

2.1. Manutenção de Software

A manutenção de software pode ser definida como a atividade de alteração de um produto de software após o momento que ele torna-se operacional. A partir deste momento, é provável que o software venha a sofrer inúmeras manutenções no decorrer de sua vida útil, seja para correção de defeitos gerados na fase de desenvolvimento, inclusão de novas funcionalidades, atendimento de requisitos de negócios não previstos na fase de desenvolvimento ou que surgiram para atender novas necessidades, ou adaptação para utilização de arquiteturas diferentes das quais foi projetado.

Dentro do ciclo de desenvolvimento de software que é, resumidamente, constituído das etapas de entendimento das necessidades, concepção, construção, operação e manutenção, esta

última etapa pode ser considerada a atividade que consome mais recursos dentro do processo de desenvolvimento, podendo demandar mais de 70% do esforço gasto (Pressman, 2002, p.786).

Pressman (2002) define 4 tipos de atividades de manutenção: corretiva, adaptativa, perfectiva e preventiva.

2.1.1. Manutenção Corretiva

A manutenção corretiva trata da atividade de consertar erros de execução que não foram encontrados na atividade de teste. Souza e Arakaki (2004) definem este tipo de manutenção como a necessidade de implantação de mudanças devido aos erros não encontrados. Estes erros possuem as seguintes classificações:

- Falhas de processamento;
- Saídas não esperadas e erros de cálculo;
- Violação de metodologias de programação (inconsistências);
- Término normal da aplicação;
- Desempenho abaixo do esperado e outros.

2.1.2. Manutenção Adaptativa

Trocas de ambiente de hardware, atualizações de sistema operacional ou sistema gerenciador de banco de dados, ou atualização de componentes de software obtidos de terceiros podem levar a incompatibilidades do sistema, necessitando da realização de manutenção. A manutenção adaptativa considera as atividades de traduzir um sistema de software a um novo contexto, conforme os exemplos citados.

Segundo Souza e Arakaki (2004), dois tipos de manutenção adaptativa devem ser considerados:

a) Mudança de dados: provocadas por alteração na estrutura dos dados, tendo como exemplo mais famoso o *bug* do ano 2000 onde foi necessário adaptação de sistemas baseados em dois dígitos do ano.

b) Mudança de processamento: onde são realizadas modificações no software para se adaptar a ambientes operacionais.

2.1.3. Manutenção Perfectiva

Ao passar dos anos, novas gerações de hardware são introduzidas no mercado, tornando obsoletas as gerações anteriores. No caso do software a situação é diferente: um sistema pode permanecer por décadas operando. Em conseqüência disso, é de se esperar que novas funcionalidades sejam agregadas ao sistema, visando aumentar sua utilidade. Dessas atividades, surge a manutenção perfectiva, cujo principal objetivo é obter um avanço qualitativo do sistema onde ela é aplicada. Cânfora et al. (1997) definem este processo como a etapa onde as alterações são solicitadas pelos usuários envolvendo inclusão, exclusão,

extensão e modificação de funcionalidades, além de documentação e melhoria de desempenho.

2.1.4. Manutenção Preventiva

A manutenção preventiva lida com os problemas antes que eles ocorram. É definida por Souza e Arakaki (2004) como sendo uma iniciativa que visa melhorar a confiabilidade e a manutebilidade futura, ou como ferramenta para obter uma nova base mais confiável para futuras ampliações. Empiricamente é um tipo de manutenção que é evitado, pois vai contra o princípio "se está funcionando, não mexa".

2.2. Evolução de Software

A dinâmica do processo de manutenção tem sido estudada por muitos autores. Em Pfleeger (2004), as leis de Lehman, sobre a evolução dos sistemas são destacadas:

1) Lei da mudança contínua: os sistemas sempre estão incompletos e para atender os requisitos do negócio devem ser modificados continuamente para continuarem úteis;

2) Lei do aumento da complexidade: com essas constantes mudanças os sistemas tornam-se cada vez mais complexos a não ser que se faça um trabalho para impedir que isto aconteça.

3) Lei fundamental da evolução do programa: os sistemas de software apresentam comportamentos e tendências que podem ser medidos e previstos

4) Lei da conservação da estabilidade organizacional: em algum momento, os recursos e serviços encontram um ponto ótimo, onde alterar os recursos não alterará significativamente os resultados;

5) Lei da conservação da familiaridade: após um determinado tempo, os efeitos de versões subseqüentes fazem pouca diferença nas funcionalidades do sistema.

2.3. Conclusões do Capítulo

Este capítulo apresentou os principais aspectos relacionados às atividades de modificação da lei de evolução de software. O conhecimento destes aspectos permite melhorar as ações quanto ao planejamento e a melhoria de qualidade, incorrendo em uma melhor confiabilidade e manutebilidade.

O processo de manutenção e evolução pode ser considerado estratégico no ciclo de vida de um software. É através das manutenções que novas funcionalidades são incluídas, migrações são realizadas ou melhorias são conquistadas. A maneira como estas alterações são realizadas aliadas a uma melhor compreensão do processo de evolução do software podem dar vantagens competitivas para as empresas que as realizam.

CAPÍTULO 3

REENGENHARIA

Objetivo do Capítulo

O objetivo deste capítulo é descrever como a reengenharia ajuda na compreensão e na evolução do sistema permitindo maior robustez, melhoria de qualidade e do desenvolvimento de software. Além disso, ele apresenta as principais iniciativas de migração de sistemas legados funcionais para o paradigma da orientação a objetos.

Este capítulo foi desenvolvido com base no estudo dos seguintes trabalhos: Magalhães e Arakaki (2005); Souza e Arakaki (2004); Ducasse (2003); Gjörwell et al. (2002); Panas et al. (2003); Ducasse et al. (1999); Warren e Ransom (2002); Bodhuin et al (2002); Ying (2004).

3.1. Reengenharia

A natureza evolutiva dos sistemas de informação requer que estes sistemas possam ser adaptados continuamente, seja para inclusão de novas funções ou para a correção de erros através de manutenções. A reengenharia é uma ferramenta utilizada no processo de manutenção com a finalidade de transformar um sistema após seu entendimento.

Segundo Ducasse et al. (1999), a reengenharia é a modificação de algum sistema a partir da engenharia reversa onde, em primeiro lugar, é recuperada uma visão de arquitetura em um nível de abstração mais alto para, em seguida, efetuar alterações nesta visão e, finalmente, implementar mudanças no código.

Para Gjörwell et al. (2002), a reengenharia preocupa-se com a análise do projeto, a implementação de sistemas legados, a aplicação de diferentes técnicas e métodos de re-projeto e a transformação para que os sistemas possam ter uma melhor arquitetura.

A reengenharia não é uma atividade trivial. Por ser uma tarefa que envolve várias fases, dentre elas o reconhecimento ou estudo do sistema legado, a remodelagem e a implementação. Seus responsáveis devem ter conhecimentos de quais tarefas envolvem a mesma, além de possuir acesso ao sistema legado. Também deve ser levado em conta que sistemas legados frequentemente estão em pleno funcionamento e a tarefa de reengenharia deve se preocupar em separar as visões que fazem sentido serem transformadas. Segundo Souza e Arakaki (2004), estas decisões devem ser feitas com prudência, pois o processo como um todo deve gerar um resultado cuja transição ocorra de maneira mais transparente possível.

3.1.1. Engenharia Reversa

A engenharia reversa é uma disciplina ou processo que permite obter um modelo de projeto de alto nível de abstração a partir do produto final. Para obter esse modelo, a engenharia reversa de software foca no código fonte e na infra-estrutura e se utiliza de técnicas de modelagem e ferramentas de software.

Para Panas et al. (2003), a engenharia reversa compreende os três sub-processos relacionados a seguir:

1) Análise do programa: etapa que foca na coleta de informações de arquitetura, de implementação e de componentes internos. A partir desta análise de baixo nível (código fonte, base de dados, etc.) são obtidos modelos de alto nível (diagrama de

classes, diagrama entidade-relacionamento, diagrama de fluxo de dados, visões de arquitetura).

2) Foco em informações: esta etapa consiste em lapidar as informações obtidas na etapa anterior. Esta lapidação consiste em: abstração da informação, onde as informações irrelevantes são subtraídas; compressão da informação, onde as informações irrelevantes são separadas do núcleo do sistema; fusão da informação, que propõe a fusão de artefatos semelhantes.

3) Visualização do programa: etapa que utiliza ferramentas de visualização gráfica para exame do programa nos níveis de código fonte, classes e arquitetura.

A realização do processo de engenharia reversa pode permitir a identificação de oportunidades de:

1) Desacoplamento: separação do software em partes menores, passíveis de serem testadas, entregues e comercializadas em partes separadas;

2) Melhoria de desempenho (tempo de resposta) da aplicação;

3) Adaptação da plataforma: transformação da implementação atual para implementação em infra-estrutura diferente;

4) Maior facilidade de manutenção: através da identificação de elementos que podem ter seu tamanho e complexidade reduzida.

3.1.2. Sistemas Legados

Pode-se considerar um sistema como legado a partir do momento que o mesmo começa a ser utilizado. Segundo Ducasse (2003), um sistema é considerado legado quando ele é

herdado de outra equipe de desenvolvimento e ainda possui valor para quem o utiliza. Se sistemas desenvolvidos em COBOL, FORTRAN ou *Assembly* são considerados como tipicamente legados, aplicativos desenvolvidos em C++, ASP e Java também podem ser considerados legados. Mesmo a manutenção de sistemas que utilizam linguagens orientadas a objetos requer constantes investimentos para controlar a entropia intrínseca de qualquer processo de manutenção de software, apesar de essas linguagens possuírem características como encapsulamento e flexibilidade. A lei da entropia sentencia que mesmo quando um sistema de software é iniciado com um planejamento e processo de desenvolvimento adequado, sua estrutura original tende a desaparecer em decorrência de novos requisitos e manutenções não previstas no início do projeto.

Em "*Renaissance of Legacy Systems*" (1998), o sistema legado é definido como um sistema antigo que permanece em operação em uma organização, caracterizando-se pelo uso de tecnologias e práticas de desenvolvimento antiquadas e tempo de vida extenso, incorrendo em grande quantidade de mudanças.

Segundo Warren e Ransom (2002), um sistema legado se caracteriza por ser bem antigo, mas que ainda está em operação dentro da organização. Muitos destes sistemas são críticos para o negócio da organização e caros de se manter. As razões para esse alto custo de manutenção incluem utilização de práticas imaturas de engenharia de software durante a sua construção, sacrifício da sua manutebilidade para satisfazer outras restrições de projeto.

3.1.3. Migração de sistemas funcionais para sistemas orientados a objeto

Partindo da premissa que sistemas orientados a objetos são mais baratos de manter, mais atualizados quanto a metodologias de desenvolvimento e mais adaptados a novas tecnologias, compilamos algumas iniciativas de migração de software com mudança de paradigma funcional para orientado a objetos. Dentre essas iniciativas destacamos as seguintes:

- De Lucia et al. (1997) propõem um processo de migração de sistemas legados para orientado a objetos envolvendo as seguintes etapas:
    - Análise estática do código legado: extração de informações relevantes do código fonte e do banco de dados;
    - Decomposição de programas "não batch": identificação de componentes de gerenciamento de interface com o usuário e de componentes do domínio de aplicação;
    - Abstração de um modelo orientado a objetos: dados persistentes são candidatos a implementar classes, enquanto blocos de código são candidatos a implementar métodos;
    - Reengenharia do sistema de acordo com os resultados das etapas de abstração de um modelo OO e decomposição: associação de métodos a classes;
    - Encapsulamento dos objetos identificados em *wrappers*: um programa OO aciona o módulo decomposto no próprio paradigma procedural;

- Conversão incremental dos objetos *wrappers:* conversão dos módulos decompostos em linguagem orientada a objetos.

- Bodhuin et al. (2002) propõem um processo de migração de sistemas on-line desenvolvidos na plataforma CICS/COBOL para WEB usando padrão de projeto MVC (Model, View, Control), baseado em uma ferramenta de extração de código que prevê as seguintes etapas:
  - Reestruturação dos comandos CICS dentro do programa COBOL;
  - Eliminação de comandos que dificultam a estruturação do código: como exemplos, os comandos GO TO e DEPENDING ON;
  - Análise estática: para identificar porções de código que possam representar conceitos do domínio da aplicação (classes);
  - Decomposição de código: para identificar componentes que refletem a lógica do negócio e futuras consistências a serem incluídas no lado cliente;
  - Extração do modelo objeto: obtido a partir da engenharia reversa do código fonte, modificado nas etapas anteriores;
  - Reengenharia: encapsulamento de funções identificadas previamente como interface com o usuário e métodos de objetos;
  - Encapsulamento (*wrapping*): dos programas COBOL, modificados na etapa anterior, por uma linguagem orientada a objetos;

- Transformação da linguagem: conversão do programa encapsulado para uma linguagem orientada a objetos.

Além das etapas descritas anteriormente, os autores criaram uma ferramenta (figura 3.1) para implementar esse processo. O analisador executa a análise estática do código fonte e o fatiador decompõe o código fonte em blocos. Essas informações são armazenadas no repositório de dados a partir delas são gerados gráficos de apoio ao engenheiro pelo controlador, a abstração do modelo OO (modelador OO), a conversão do código legado, assim como o encapsulamento e re-implementação da interface com o usuário.

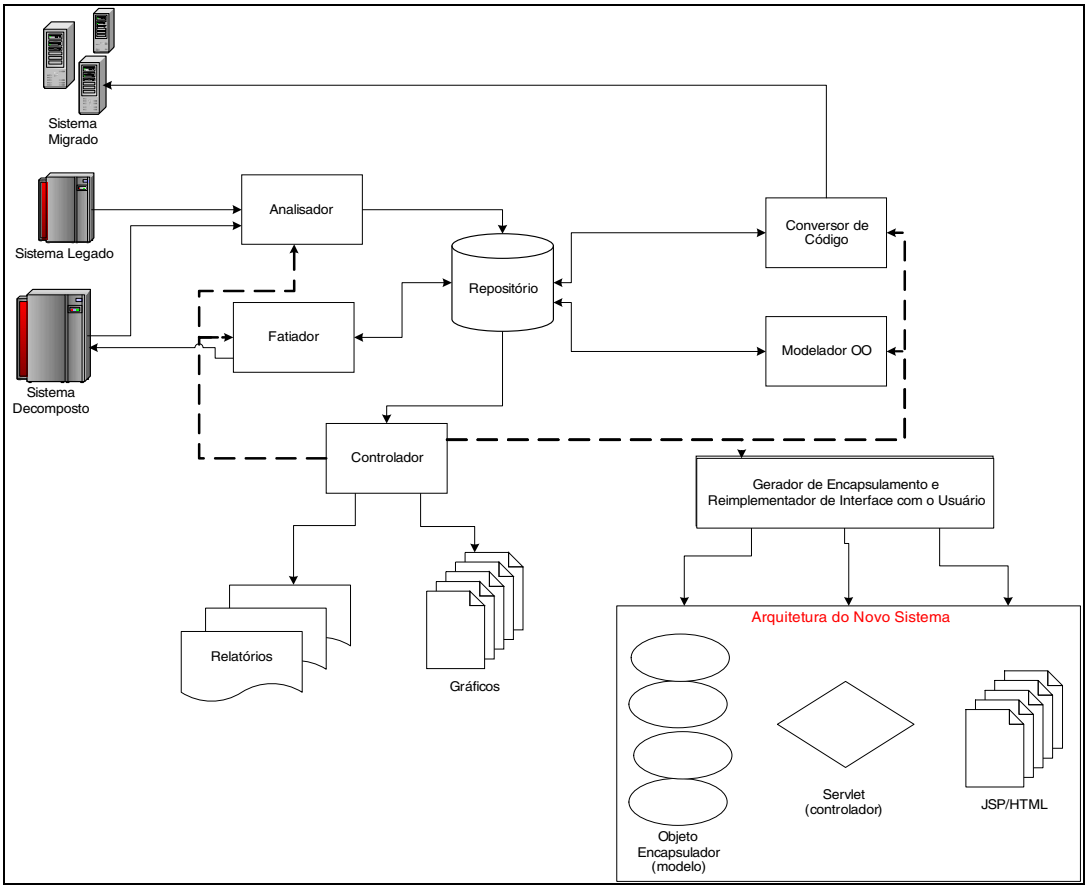

Figura 3.1 – Automação de Migração de Código legado (Bodhuin et al., 2002, p.3).

- O modelo de transição de estados de Ying (2004) que permite que os requisitos de qualidade, manutebilidade e reusabilidade sejam modelados como se fossem os objetivos do software. Esse processo propõe as seguintes etapas:
    - O código fonte original é analisado sintaticamente e representado em termos de árvore de sintaxe abstrata (Abstract Sintax Tree – AST);
    - A árvore sintática abstrata é decomposta em vários blocos. Cada bloco pode incluir extensivas porções de código com características relacionadas a uma classe candidata;
    - Os blocos de código fonte são convertidos em uma linguagem genérica formada a partir de várias linguagens procedurais;
    - Os requisitos de qualidade de mais alto nível são eliciados com base em conhecimento, entrevistas com os usuários e documentos. O ponto focal é traduzir os requisitos de qualidade de alto nível em decisões de projeto e características de código fonte que possam ser mensuradas. O processo de refinamento é modelado usando grafos de interdependência de objetivos de software;
    - Um processo de transformação é realizado em cada bloco, no qual é aplicado um processo de migração direcionado por requisitos de qualidade. Cada bloco a ser transformado é inserido com entrada no processo de migração, onde os blocos migrados anteriormente já fazem parte do novo sistema;

- O processo de avaliação de qualidade seleciona o modelo objeto com a melhor qualidade baseado no algoritmo de Viterbi e computa a probabilidade de atingir o requisito de qualidade;
- Uma vez que todos os blocos foram migrados, o modelo objeto está consolidado com a melhor qualidade possível.

- Em Cânfora et al. (2006) tem-se a proposta de migração de um sistema *web* baseado em formulário para um sistema baseado em serviços (*web services*). Esse processo é voltado à conversão de programas on-line e diferencia-se por não utilizar o código fonte como base para conversão (modelo caixa branca). Ele baseia-se no modelo caixa preta, onde há a utilização de cenários de casos de uso para auxiliar a compreensão do comportamento de código e a obtenção dos dados obtidos a partir da interface com o usuário. Esse processo possui as seguintes fases:
  - Identificação: as interfaces com o usuário do sistema legado são encaradas como cenários de um caso de uso. Para cada cenário, as telas retornadas pela aplicação legada a cada interação e as ações realizadas pelos usuários nos campos das telas são armazenadas. Telas com layout similar são agrupadas e associadas a um mesmo modelo de tela. Cada interação entre o usuário e a aplicação será associada ao modelo de tela. Assim sendo, o autômato de estado finito (compreendendo, estados de interação, transições e ações disparadas pelas transições) é obtido. No final desta fase, um conjunto de entradas providas pelos usuários e o

conjunto de saídas retornadas pela aplicação em cada cenário do caso de uso será definida;

- Projeto: nessa fase, para que o autômato posse ser interpretado, todas as variáveis são identificadas e cada estado de interação do autômato de estados finitos deve ser associado a um conjunto de ações a serem realizadas sobre as variáveis ou campos das telas. O modelo autômato é convertido em um arquivo XML e armazenado em repositório de dados;
- Implementação: todas as operações necessárias para publicar o *web services* e exportá-las para um servidor de aplicação são realizadas.
- Validação: um teste é realizado para validar o serviço encapsulado

3.2. Conclusões do Capítulo

A reengenharia envolve várias fases, dentre elas o reconhecimento ou estudo do sistema legado, a remodelagem e a implementação. Os responsáveis pela execução desta tarefa devem ter conhecimento de quais tarefas envolvem a mesma, além de possuir acesso ao sistema legado. Também devem levar em conta que sistemas legados frequentemente estão em pleno funcionamento e a tarefa de reengenharia deve se preocupar em separar as visões que fazem sentido serem transformadas. Quando há mudança de paradigma de desenvolvimento, esta atividade deve ser exercida com mais cuidado, pois o processo como um todo deve gerar um resultado cuja transição ocorra de maneira mais transparente possível.

CAPÍTULO 4

QUALIDADE DE SOFTWARE

Objetivo do Capítulo

O objetivo deste capítulo é descrever os conceitos que sustentam as práticas e técnicas da avaliação da qualidade de software que serão aplicadas nos capítulos seguintes deste trabalho, fornecendo subsídios teóricos sobre sua utilização. São apresentadas as características e sub-características e algumas visões dos interessados no processo.

Este capítulo foi desenvolvido com base no estudo dos seguintes trabalhos: Magalhães e Arakaki (2005); Souza e Arakaki (2004); Pressman (2002); NRB ISO/IEC 9126 (2003); NBR ISO/IEC 14598-1 (2001); Bosch e Molin (1999).

4.1. Qualidade de Software

Segundo a norma NBR ISO/IEC 14598-1 (2001), qualidade é a totalidade de características de uma entidade que lhe confere a capacidade de satisfazer as necessidades explícitas e implícitas (embora não tenham sido explicitadas, devem estar presentes em determinadas condições de uso). Entende-se necessidade como a expectativa quanto aos efeitos do uso do produto. Para Pressman (2002), qualidade é uma mistura complexa de fatores que sofrerão variações de acordo com a aplicação e respectivos usuários. Ele enfatiza três pontos sobre a qualidade: (1) Requisitos de software compõem a base sobre a qual a qualidade é medida; (2) Padrões específicos definem um conjunto de critérios de desenvolvimento que guiam a maneira pela qual o software é construído e; (3) Existe um

conjunto de requisitos implícitos que frequentemente não são mencionados, mas comprometem a qualidade de software (exemplo: facilidade de uso).

As normas de qualidade direcionam para um trabalho que permite que a interpretação sobre os diversos aspectos da qualidade seja feita igualmente pelos diversos usuários da norma, conduzindo a um caminho único que torna possível a comparação de resultados. A força da norma consiste em traçar parâmetros gerais que todos possam seguir.

A norma ISO 9001 é a mais adequada para desenvolvimento e manutenção de software. A norma ISO 9000-3 traz processos para aplicar a ISO 9001 nas áreas de desenvolvimento, fornecimento e manutenção de software e para criar um sistema de qualidade de software.

Para avaliação do produto de software existe a norma NBR ISO/IEC 14598. A avaliação de qualidade do software é um processo de se obter, de maneira correta, sistematizada e clara, o nível de qualidade do sistema sobre determinada característica. Os seguintes passos devem ser realizados em um processo de avaliação:

a) Estabelecimento dos requisitos de avaliação: onde será estabelecido o propósito da avaliação, identificado tipos e produtos a serem avaliados, e onde será especificado o modelo de qualidade;

b) Especificação da avaliação: envolve seleção de medidas, estabelecimento de níveis de pontuação e de critérios de julgamento;

c) Projeto da avaliação: neste passo é produzido o plano de avaliação;

d) Execução da avaliação: envolve obtenção de medidas, comparação com a utilização de critérios e o julgamento dos resultados.

A norma NBR ISO/IEC 9126 – Qualidade de Produto de Software – foi desenvolvida para identificar atributos-chave de qualidade de software (Pressman 2002). Deve ser utilizada na definição de requisitos de qualidade, avaliação das especificações para verificar se os requisitos de qualidade estão sendo atendidos e para avaliar o software antes da entrega ao cliente. Como a norma não define métricas, cada empresa ou instituição deve criar as suas próprias.

A figura 4.1 apresenta a estrutura da norma NBR ISO/IEC 9126 cujas definições são detalhadas a seguir:

a) Qualidade de uso: é a capacidade do produto de software permitir que cada usuário atinja metas especificadas com eficácia, produtividade, segurança e satisfação em contextos de uso determinados. Varia de acordo com o contexto de uso do *software;*

b) Qualidade externa: é a capacidade de um conjunto de atributos externos do software satisfazer necessidades explícitas e implícitas quando o software for utilizado em condições especificadas

c) Qualidade interna: é a capacidade de um conjunto de atributos internos do software ser utilizado em condições especificadas. Há dois tipos de requisitos de qualidade interna: qualidade de projeto e qualidade de código.

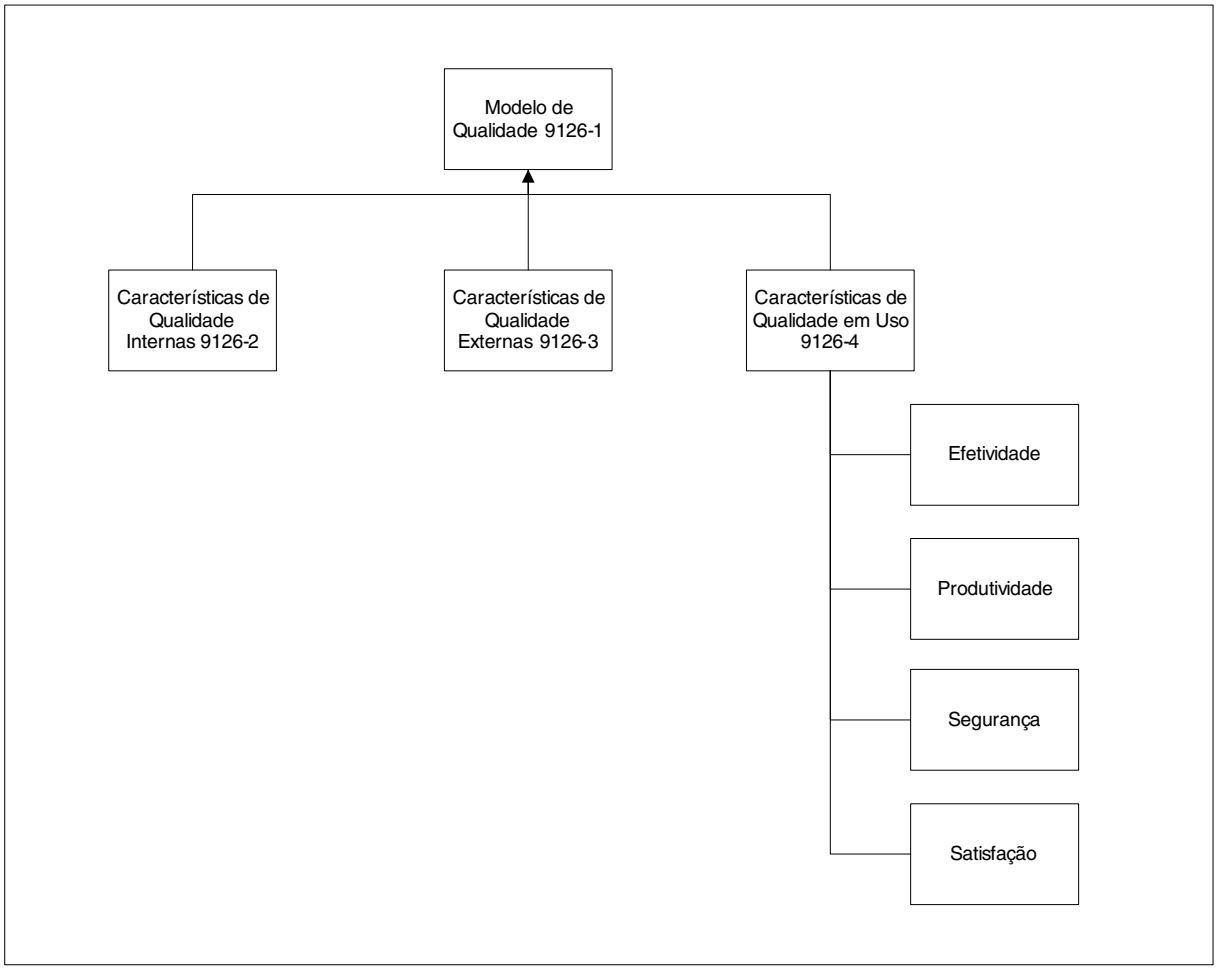

Figura 4.1 – Estrutura da Norma ISO/IEC 9126 (2003).

A série de normas NBR ISO/IEC 9126 define seis características, com um mínimo de sobreposição, que podem ser utilizadas para se avaliar a qualidade do produto de *software*. Estas características são:

- Funcionalidade;
- Confiabilidade;
- Eficiência;
- Manutebilidade;
- Portabilidade;
- Usabilidade.

Cada uma destas características está descrita em mais detalhes no quadro 4.1, que também mostra as respectivas sub-características. Fica a cargo dos usuários estabelecer as métricas de avaliação de cada item ou subitem.

Característica	Sub-característica	Definição
Funcionalidade Conjunto de atributos que evidenciam a existência de um conjunto de funções e suas propriedades especificadas	Adequação	Atributos de software que evidenciam a presença de um conjunto de funções e sua apropriação para as tarefas especificadas.
	Acurácia	Atributos de software que evidenciam a geração de resultados ou efeitos corretos ou conforme acordados.
	Interoperabilidade	Atributos de software que evidenciam sua capacidade de interagir com sistemas especificados.
	Segurança de Acesso	Atributos de software que evidenciam sua capacidade de evitar acesso não autorizado, acidental ou deliberado a programas e dados.
Confiabilidade Conjunto de atributos que evidenciam a capacidade do *software* manter o seu nível de desempenho sob condições estabelecidas durante um período de tempo estabelecido.	Maturidade	Atributos de software que evidenciam a freqüência de falhas por defeitos de software.
	Tolerância a falhas	Atributos de software que evidenciam a sua capacidade em manter um nível de desempenho especificado nos casos de falhas no software ou de violação nas interfaces especificadas.
	Recuperabilidade	Atributos de software que evidenciam sua capacidade de restabelecer seu nível de desempenho especificado nos casos de falhas no software ou de violação nas interfaces especificadas.
Usabilidade Conjunto de atributos que evidenciam o esforço necessário para poder-se utilizar o *software,* bem como o julgamento individual deste uso, por um conjunto implícito ou explícito de usuários.	Intelegibilidade	Atributos de software que evidenciam o esforço do usuário para reconhecer o conceito lógico e sua aplicabilidade.
	Apreensibilidade	Atributos de software que evidenciam o esforço do usuário para apreender sua aplicação.
	Atratividade	Atributos de software que evidenciam o quanto ele é atrativo ao usuário.
	Operabilidade	Atributos de software que evidenciam a facilidade de sua operação.

Quadro 4.1 - Norma NBR ISSO/IEC 9126: Caraterísticas e Subcaracterísticas (2003).

Característica	Sub-característica	Definição
Eficiência Conjunto de atributos que evidenciam o relacionamento entre o nível de desempenho do software e a quantidade de recursos usados sob condições estabelecidas.	Comportamento em relação ao tempo	Atributos de software que evidenciam seu tempo de resposta, tempo de processamento e velocidade na execução de suas funções.
	Comportamento em relação aos recursos	Atributos de software que evidenciam a quantidade de recursos usados e a duração de seu uso na execução de suas funções.
Portabilidade Conjunto de atributos que evidenciam a capacidade do software em ser transferido de um ambiente para outro.	Adaptabilidade	Atributos de software que evidenciam sua capacidade de ser adaptado a ambientes diferentes especificados, sem a necessidade de aplicação de outras ações ou meios além daqueles fornecidos para esta finalidade pelo software considerado.
	Capacidade para ser instalado	Atributos de software que evidenciam o esforço necessário para sua instalação num ambiente especificado.
	Coexistência	Atributos de software que evidenciam a capacidade de coexistir com outros softwares sem alterações.
	Capacidade para substituir	Atributos de software que evidenciam sua capacidade e esforços necessários para substituir um outro software, no ambiente estabelecido para este outro software.
Manutebilidade Conjunto de atributos que evidenciam o esforço necessário para fazer modificações especificadas no software.	Analisabilidade	Atributos de software que evidenciam o esforço necessário para diagnosticar deficiências ou causas de falhas, ou para identificar partes a serem modificadas.
	Modificabilidade	Atributos de software que evidenciam o esforço necessário para modificá-lo, remover seus defeitos ou adaptá-lo a mudanças ambientais.
	Estabilidade	Atributos de software que evidenciam o risco de efeitos inesperados ocasionados por modificações.
	Testabilidade	Atributos de software que evidenciam o esforço necessário para validar o software modificado.
Todas Para todas as características de qualidade citadas acima, deve-se também considerar os aspectos de conformidade.	Conformidade	Atributos de software que fazem com que ele esteja de acordo com as normas, convenções ou regulamentações previstas em leis e descrições similares, relacionadas à aplicação.

Quadro 4.1 - Norma NBR ISSO/IEC 9126: Características e Sub-características (continuação)

4.2. Visões da Qualidade de Software

Neste item serão apresentadas algumas visões nas quais a qualidade de software pode ser avaliada. As visões apresentadas são:

- Visão do usuário: foca principalmente no uso do software, sua eficiência e nos efeitos gerados pelo software, sem se interessar pelos aspectos internos ou como o software foi desenvolvido;
- Visão do desenvolvedor: está interessado na qualidade do software nas fases final e intermediárias das atividades de desenvolvimento e de manutenção. Preocupa-se, mesmo que de forma não organizada, com aspectos internos (confecção do código, arquitetura, etc.) e externos ao software (visão do usuário);
- Visão do gerente: está mais preocupado com a qualidade geral do software e com a melhoria da qualidade porque isto pode incorrer em redução de custo.

4.3. Métrica de Software

"Métrica é a medida quantitativa do grau ao qual o sistema, componente ou processo possui de um determinado atributo". Isto segundo a definição do IEEE *Standard Glossary of Software Engineering Terms* citado em Pressman (2002). Exemplo: número médio de erros encontrados por revisão. Também é definida como um método e uma escala quantitativa que podem ser usados para determinar o valor que uma particularidade recebe em um problema de software específico.

4.4. Conclusões do Capítulo

Neste capítulo foram descritas os principais aspectos de qualidade, medição e das normas que podem ser utilizadas para a mensuração de características do software.

As normas de qualidade direcionam para um trabalho que permite que a interpretação sobre os diversos aspectos da qualidade seja feita igualmente pelos diversos usuários da norma, conduzindo a um caminho único que torna possível a comparação de resultados. A força da norma consiste em traçar parâmetros gerais que todos possam seguir.

CAPÍTULO 5

ARQUITETURA DE SOFTWARE

Objetivo do Capítulo

O objetivo deste capítulo é descrever os conceitos que sustentam as práticas e técnicas de arquitetura de software que serão utilizadas nos capítulos seguintes, fornecendo subsídios teóricos sobre sua aplicação. São apresentados os tipos de visão e estilos arquiteturais e a importância da arquitetura de software para atingir requisitos de qualidade.

Este capítulo foi desenvolvido com base no estudo dos seguintes trabalhos: Magalhães e Arakaki (2005); Souza e Arakaki (2004); Ducasse (2003); Gjörwell et al. (2002); Panas et al. (2003); Ducasse et al. (1999); Clements et al. (2003); Bass et al. (2003); Bachmann et al. (2005).

5.1. Arquitetura de Software

A arquitetura de software (Clements et al., 2003) é a estrutura ou as estruturas do sistema de computador. Ela define o sistema em termos de componentes e interação entre estes componentes (clientes, servidores, banco de dados, filtros e camadas).

Dado que a manutenção e a evolução de um produto de software é uma conseqüência natural da evolução dos negócios e do ambiente operacional, é compreensível que o software fique cada vez maior e mais complexo. Para ajudar a lidar com esse crescimento e aumento de complexidade, a arquitetura de software fornece um nível de abstração que facilita o tratamento de questões relacionadas a manutebilidade, tolerância a falhas, eficiência, acurácia,

interoperabilidade e portabilidade. A arquitetura pode orientar a manutenção dos sistemas nos casos de:

1) Servir como base de conhecimento: no caso de aplicativos grandes, por exemplo, muitos podem ser os profissionais envolvidos, onde cada um deles terá um enfoque específico. Modelos de representação arquitetural geram as visões necessárias para permitir uma melhor integração desses diferentes profissionais;
2) Proporcionar o correto crescimento e evolução: uma arquitetura bem definida pode tornar mais fácil trocar, corrigir, evoluir, integrar o sistema ou partes dele;
3) Integração com outros sistemas: isso pode ser obtido ao se deixar claro os pontos e as formas de interface;
4) Otimização: para manter o sistema enxuto e eficiente, ou seja, sem funcionalidades repetidas, mal implementadas ou não utilizadas;
5) Correção: para ajudar a localizar mais facilmente erros de programa, entender as suas várias conseqüências e formas de correção.

As alternativas de arquitetura são várias. Cada uma delas prioriza certos aspectos em detrimento de outros. Não existe uma arquitetura perfeita para todas as situações. Segundo Clements et al. (2003) a arquitetura pode ser baseada em tipos de visão, que podem ser entendidos como a especificação do tipo de informação que ela irá prover, a saber: módulos; componentes e conectores; e alocação. O quadro 5.1 (adaptado de Clements et al., 2003) mostra essas informações com um pouco mais de detalhes.

Tipo de Visão	Estilo Arquitetural	Descrição
Módulos Mostra a estrutura do sistema como um conjunto de unidades de implementação. Cada unidade possui um conjunto de responsabilidades. Tem por finalidade a comunicação entre os envolvidos e ser base para análise.	Decomposição	Explora a relação: é parte de
	Uso	Destaca a dependência funcional, explorando a relação "depende de".
	Generalização	Mostra o relacionamento de especialização entre os módulos: relação hierárquica "*is a*"
	Camada	Mostra as unidades de implementação organizadas em camadas
Componentes e Conectores Exprimem o comportamento do sistema em execução	Canos e Filtros	Mostra as transformações sucessivas em dados processados
	Dados compartilhados	Compartilha dados entre componentes
	Publicador-Subscritor	Componentes anunciam e tratam os eventos
	Cliente Servidor	Componentes interagem requisitando serviços de outros componentes
	Peer-to-Peer	Componentes interagem entre si requisitando serviços ou respondendo a solicitações (sem assimetria)
	Processos Comunicantes	Componentes executam tarefas de forma distribuída e concorrente através de mecanismos de conexão
Alocação Mapeia os elementos de software nos ambientes	Distribuição	Mapeia os elementos de software no ambiente de hardware
	Implementação	Mapeia os elementos nos sistema de arquivos e estrutura de desenvolvimento
	Atribuição de Trabalho	Mapeia os elementos de software nas equipes de desenvolvimento

Quadro 5.1 - Visões e estilos arquiteturais (Clements et al., 2003, p.69-217).

5.2. Tipos de Visão Módulos

Esse tipo de visão mostra a estrutura do sistema como um conjunto de unidades de implementação. Hoje em dia, a forma na qual o sistema é decomposto em unidades manuseáveis permanece como uma das mais importantes formas de estruturação por que permite determinar como o código fonte está organizado, que tipos de premissas cada parte pode fazer sobre serviços providos por outras partes e como cada parte são agregadas dentro

da aplicação como um todo. A escolha desse tipo de visão permite determinar como a alteração a ser realizada em um módulo afeta as demais partes do sistema e, consequentemente, a habilidade do sistema em atender requisitos de qualidade como funcionalidade, manutebilidade, portabilidade e reuso.

Os seguintes estilos arquiteturais são exemplos de tipos de visão módulos:

a) Decomposição – decompõe o código de forma *top-down* em módulos, sub-módulos, e assim sucessivamente. Orienta a obtenção de algum atributo de qualidade (exemplo, obter a manutebilidade quando se utiliza o encapsulamento). Pode ajudar nas decisões de comprar ou construir, além de prover suporte para o desenvolvimento de famílias de produto ao verificar semelhanças e diferenças entre os módulos. Na figura 5.1 temos que o módulo A é decomposto nos módulos B, C e D;

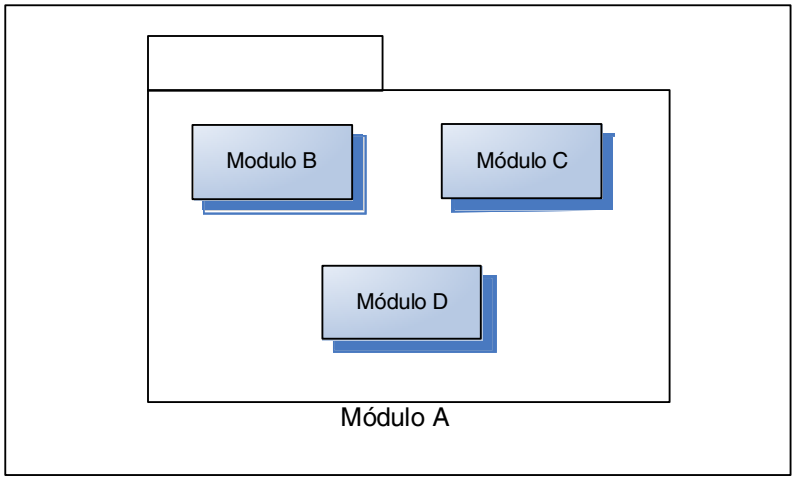

Figura 5.1 – Exemplo de estilo Decomposição

b) Uso – esse estilo (figura 5.2) explora a relação “depende de”. Informa aos desenvolvedores que módulos devem existir para que ele funcione. Esse estilo é útil

para determinar restrição de implementação, orientar o desenvolvimento incremental, depuração e testes e para avaliar os efeitos de mudanças específicas (testabilidade);

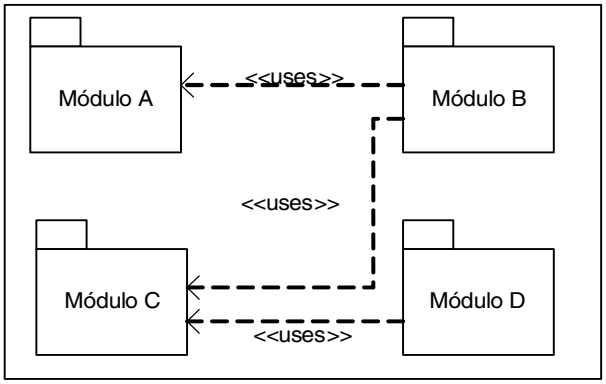

Figura 5.2 – Exemplo de estilo Uso

c) Generalização – explora a relação "is a". Suporta a evolução ou extensão de elementos de software, pois captura o que é comum e que varia entre os módulos. O elemento que generaliza o outro fornece o que é comum aos dois (estrutura, comportamento e restrições gerais). A utilização desse estilo (vide exemplo na figura 5.3) facilita o reuso e a manutenção, uma vez que é mais fácil entender o que difere um módulo do outro do que entender toda a estrutura do sistema ;

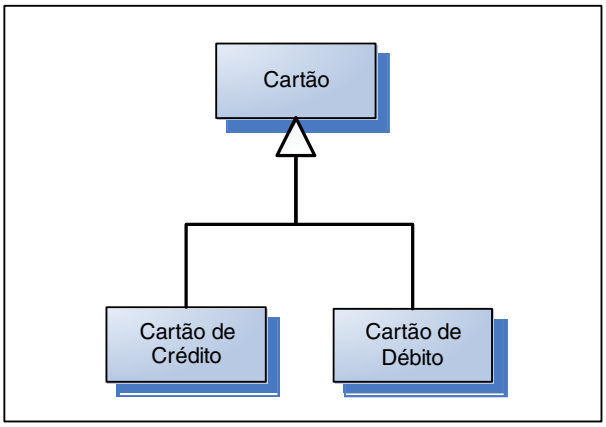

Figura 5.3 – Exemplo de estilo Generalização

d) Camadas – o estilo mostra o software dividido em camadas, onde os componentes dessa provem serviços para os componentes da camada superior. Essa forma de organização dá suporte a manutebilidade e portabilidade. No exemplo da figura 5.4, a camada mais externa representa interface com o usuário, enquanto na camada mais interna temos componentes de comunicação com o sistema operacional e nas intermediárias temos componentes utilitários e de aplicação.

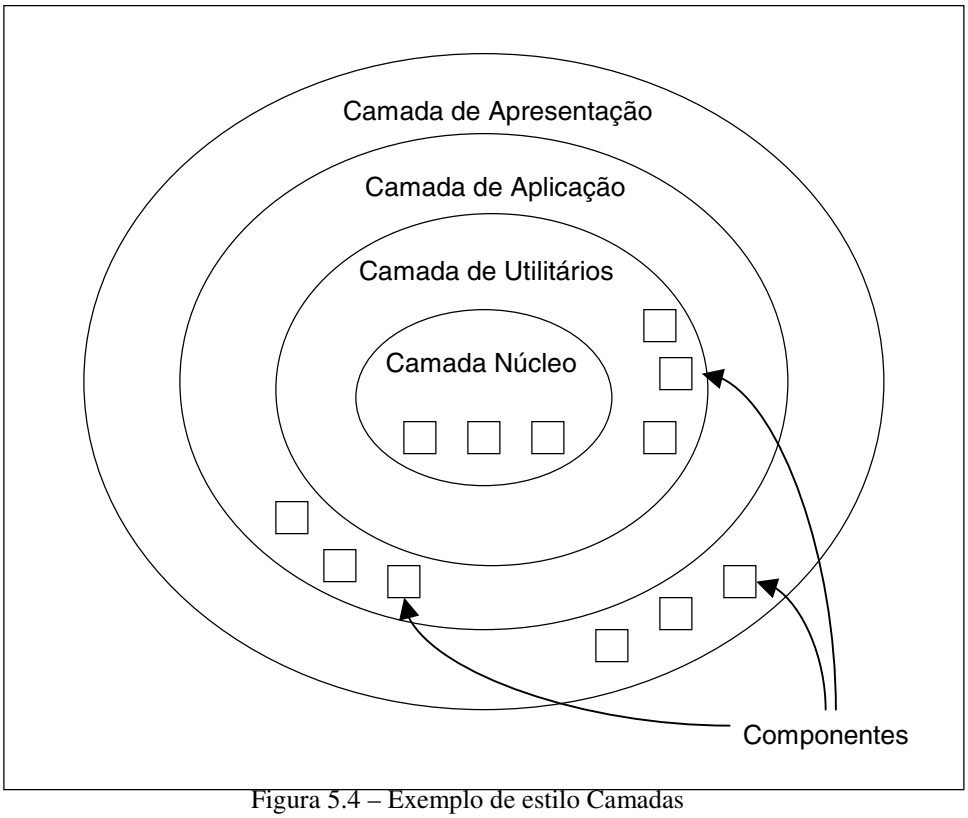

Figura 5.4 – Exemplo de estilo Camadas

Outro exemplo clássico é apresentado na figura 5.5, onde é apresentado o modelo de referência OSI/ISO para redes de computadores (cada camada é vista como um componente que pode ser implementado por hardware ou software).

A arquitetura de redes é formada de uma estrutura hierárquica composta por camadas com interfaces entre elas. Associada a cada camada há um protocolo que

define o formato, conteúdo e significado dos dados trocados entre sua parceira de mesmo nível.

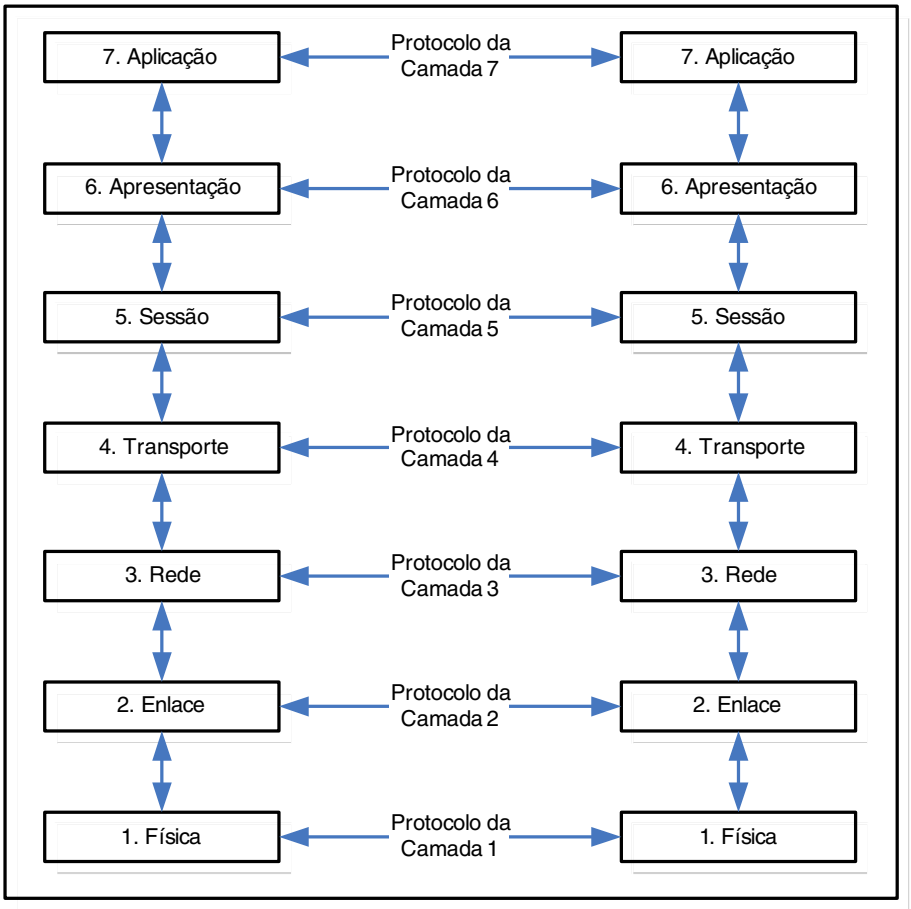

Figura 5.5 – Exemplo de estilo camadas – modelo de referência OSI/ISO

5.3. Tipos de Visão Componentes e Conectores

Os tipos de visão componentes e conectores permitem mapear os elementos existentes em tempo de execução, como processos, objetos, clientes, servidores e bases de dados, e as possíveis formas de interação entre esses elementos, tais como links de comunicação e protocolos, fluxos de informação e acessos a base de dados compartilhadas.

Usamos este tipo de visão para compreender os atributos de qualidade do sistema em execução, tais como performance, confiabilidade e disponibilidade. Ele nos ajuda a responder questões como:

- Quais são os principais elementos em execução e como eles interagem entre si?
- Quais as principais bases de dados compartilhadas?
- Quais partes do sistema são replicadas e em quantas vezes?
- Como ocorre a transformação das informações dentro do sistema em execução?
- Que protocolos são utilizados entre os elementos de comunicação?
- Que partes do sistema executam em paralelo?
- Como mudar a estrutura do sistema em tempo de execução?

Os estilos arquiteturais desse tipo de visão são os seguintes:

a) Canos e filtros: o padrão de interação desse estilo arquitetural caracteriza-se por sucessivas transformações de dados. Os dados chegam a um filtro, são transformados e passados para outros filtros e assim sucessivamente. Um filtro pode passar dados para um ou mais filtros subseqüentes. Como ele não requer conhecimento dos seus vizinhos, cada um trabalha de forma independente. Esse estilo é adequado para projetar sistemas que requerem vários estágios de processamento. A figura 5.6 apresenta um exemplo desse estilo.

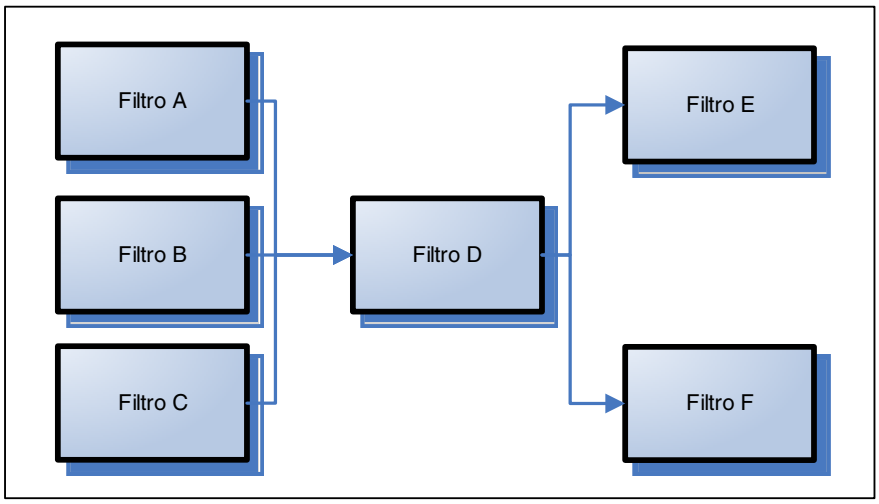

Figura 5.6 – Exemplo de estilo Canos e Filtros

b) Dados compartilhados: esse estilo é caracterizado pela troca de dados persistentes. Os dados são acessados por vários componentes que atualizam, adicionam, apagam ou realizam qualquer outro tipo de modificação e há pelo menos um banco de dados compartilhado para a retenção de dados persistentes. Se o banco de dados compartilhado informa a chegada de dados de interesse aos respectivos componentes consumidores, esse estilo é chamado de quadro negro. Se os componentes é que tem a responsabilidade de obter as informações, então esse estilo é denominado repositório. Em sistemas mais modernos é comum a coexistência das duas situações. Análises associadas a este estilo estão geralmente voltadas para performance, segurança, privacidade, confiabilidade e compatibilidade com outros bancos de dados. A figura 5.7 apresenta um exemplo desse estilo.

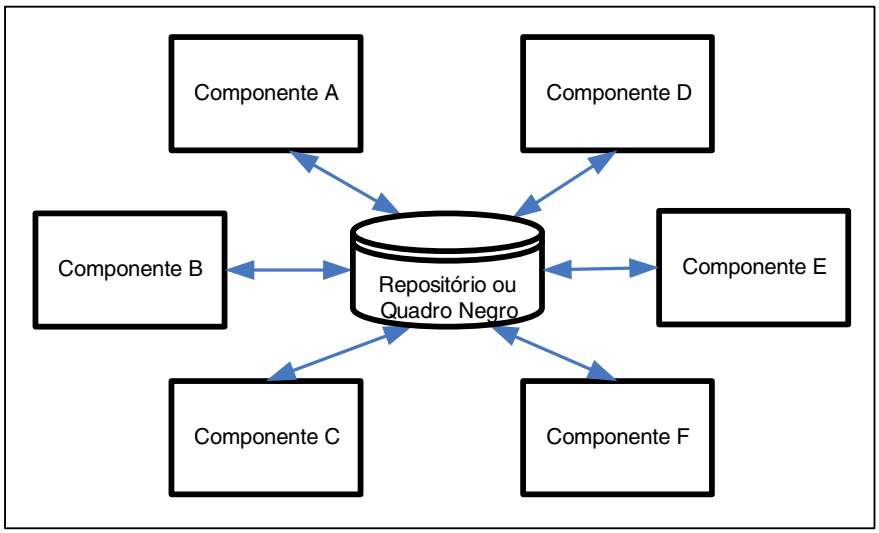

Figura 5.7 – Exemplo de estilo Dados Compartilhados

c) Publicador-Subscritor: neste estilo os componentes interagem através de eventos anunciados. Componentes podem tratar um conjunto de eventos. O trabalho da infra-estrutura de execução do publicador-subscritor é assegurar que cada evento publicado é entregue para todos os componentes que o tratam. A principal forma de conector nesse estilo é um tipo de barramento onde os componentes colocam os eventos que são entregues para os componentes que os tratam. Como esse estilo é usado para enviar mensagens para receptores desconhecidos, novos componentes receptores podem ser adicionados sem a necessidade de modificar os produtores.

d) Cliente-Servidor: os componentes interagem requisitando serviços de outros componentes. A essência desse estilo é que a comunicação é tipicamente iniciada e gerada pelo cliente. Os servidores provem um conjunto de serviços através de uma ou mais interfaces. A utilização desse estilo apresenta uma visão que separa as aplicações dos clientes dos serviços que elas usam, facilitando o entendimento através da separação de serviços comuns. O agrupamento de funcionalidade nesse estilo prove

uma base para entendimento da implementação na plataforma de hardware ou para interagir com serviços de sistemas legados. O particionamento das funcionalidades em clientes e servidores permite que eles sejam independentemente alocados em camadas, assim como o suporte a eficiência, escalabilidade e confiabilidade.

e) *Peer-to-Peer*: nesse estilo, os componentes interagem diretamente através de troca de serviços. A comunicação *peer-to-peer* é um tipo de solicitação/resposta sem a assimetria encontrada no estilo cliente-servidor, uma vez que cada componente pode interagir com qualquer outro solicitando seus serviços. A conexão neste estilo pode envolver um complexo protocolo bi-direcional de interação, refletindo a dupla comunicação que pode existir entre dois ou mais componentes. Exemplos de sistemas peer-to-peer incluem arquiteturas baseadas em infra-estruturas de objetos distribuídos como CORBA, COM+, e Java RMI. Visões arquiteturais de componentes executáveis, usando notações como diagrama de colaboração, são geralmente exemplos de representação desse estilo. A computação *peer-to-peer* é usada em aplicações de processamento distribuído. Utilizando uma implementação apropriada, a aplicação pode fazer uso eficiente de recursos de CPU e disco distribuindo tarefas de processamento intensivo através da rede de computadores e aproveitando a vantagem dos recursos disponíveis.

f) Processos comunicantes: a interação de componentes executando concorrentemente através de vários mecanismos de conexão é a característica desse estilo. Entre os exemplos de mecanismos de conexão podemos citar sincronização, passagem de mensagem, controle, troca de dados, entre outros. Processos de comunicação são comuns na maioria dos grandes sistemas e necessários em todos os sistemas

distribuídos. Esse estilo é usado para o entendimento de quais porções do sistema operam em paralelo, a inclusão de componentes dentro de processos e *threads* de controle dentro do sistema. Esse estilo pode usado para analisar performance e confiabilidade e também pode ser útil na fase de projeto, onde é decidida a atribuição de componentes a processos.

5.3. Tipos de Visão Alocação

O tipo de visão Alocação corresponde a mostrar o software em seus relacionamentos com o hardware, o sistema de arquivos e com as equipes de desenvolvimento. É através do mapeamento da arquitetura de software com o hardware, com a equipe de desenvolvimento e com o sistema de arquivos que a performance pode ser analisada, o gerenciamento das atividades do projeto pode ser realizada assim como o gerenciamento do sistema. Os estilos arquiteturais mais comuns desse tipo de visão são:

a) Distribuição: os elementos dos estilos arquiteturais componentes e conectores e processos comunicantes são alocados em plataformas de execução. As restrições para qualquer alocação particular são requisitos expressos através de elementos de software e através da forma como esses requisitos são atingidos pelas características dos elementos relevantes de hardware. Dentre as mais importantes propriedades dos elementos do estilo arquitetural distribuição, sejam eles software ou hardware, são aqueles que afetam a alocação do software aos elementos físicos. A forma como um elemento físico satisfaz um requisito de elemento de software é determinado pela propriedade de ambos. Se um elemento de software requer um espaço mínimo para ser alocado, qualquer elemento do ambiente operacional que tenha capacidade para armazenar esse espaço será candidato para alocar esse elemento de software. Dentre as

propriedades dos elementos de ambiente relevantes destacamos: CPU; disco; memória; largura de banda (capacidade de comunicação) e tolerância a falhas. Para os elementos de software são relevantes as propriedades: consumo de recursos; recursos necessários para atendimento de requisitos e restrições; nível de estabilidade. Esse estilo é utilizado para analisar performance, reuso e segurança;

b) Implementação: mapeia os elementos do estilo arquitetural módulo com a infra-estrutura de desenvolvimento. A implementação de um módulo sempre resulta em vários arquivos separados como código fonte, arquivos de definição, arquivos que descrevem como construir o executável e arquivos que são resultados da compilação de um módulo. Todos esses arquivos necessitam ser organizados de forma a não perder o controle de integridade do sistema. Esse estilo é importante para o gerenciamento das atividades de desenvolvimento e para a elaboração de processos. Desenvolvedores usam este estilo para identificar arquivos que precisam atualizar, testar ou construir, além de criar novas versões;

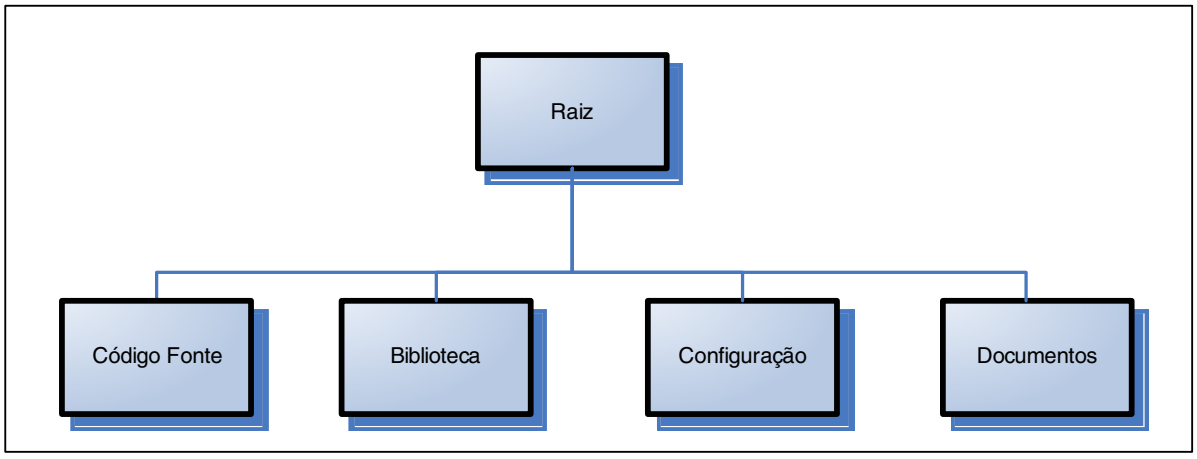

Figura 5.8 – Exemplo de estilo Implementação

c) Atribuição de trabalho: esse estilo arquitetural representa o mapeamento das equipes de desenvolvimento à arquitetura de software. Ele é um importante estilo de alocação, uma vez que pessoas não são simplesmente alocadas a um sistema em construção no final de seu desenvolvimento. Nos casos de software comprados que não necessitam ser implementados, há a necessidade de alguém que o represente, o teste e entenda como ele funciona. O mesmo ocorre para os softwares de apoio ao ambiente de desenvolvimento. A atribuição de trabalho mostra as principais unidades de software que devem estar presentes para formar o sistema a ser trabalhado e quem vai produzi-lo (quadro 5.3), assim como as ferramentas e ambientes onde o software será desenvolvido e ela é apropriada para gerenciamento de alocação de recursos, estimativa de tempo e custo.

Sistema	**Programa**	**Responsável**
Débito Automático	Programa P1	Analista A
	Programa P2	Analista B
	Programa P3	Analista B
	Programa P4	Analista A
	Programa P5	Analista B
	Programa P6	Analista A
	Programa P7	Analista A

Quadro 5.3 – Exemplo de Atribuição de trabalho

5.4. Os Estilos Arquiteturais e os atributos de qualidade de Software

Embora algumas iniciativas levem em consideração alguns aspectos de qualidade, entende-se que a qualidade de um sistema possa ser atingida de forma mais ampla. Sendo assim, uma atenção especial deve ser dada para a qualidade na elaboração da arquitetura. A seguir citamos alguns autores que reforçam essa idéia:

- Para Bosch e Molin (1999), provavelmente a atividade mais complexa durante o processo de desenvolvimento, é a transformação da especificação de requisitos em arquitetura de software. Embora outras atividades também sejam desafiadoras, elas são melhor entendidas e possuem um melhor entendimento por parte dos desenvolvedores, além de contar com um melhor suporte metodológico e tecnológico. O processo de transformação arquitetural é menos formal e mais intuitivo. A arquitetura de software tem recebido considerável atenção durante os anos recentes devido especialmente, entre outras razões, ao fato dos requisitos de qualidade terem uma forte influência na arquitetura do sistema;

- Segundo Bass et al (2003), a arquitetura é crítica para a realização de vários atributos de qualidade de um sistema e esses atributos devem ser projetados e avaliados em um nível arquitetural. A arquitetura, por si só, não atinge os atributos de qualidade, pois os detalhes de implementação como algoritmos (considerados como aspectos não arquiteturais) também devem ser levados em conta. Mas, ela provê o alicerce para se atingir os objetivos de qualidade de forma mais ampla;

- Bachmann et al. (2005), ao propor um framework que possa atingir objetivos de qualidade a partir do projeto da arquitetura de software, afirma que a arquitetura de software e os atributos de qualidade que ela permite atingir devem ser umas das tarefas que o arquiteto deve empregar maior tempo e esforço.

Nos tópicos anteriores relacionamos os tipos de visão e respectivos estilos arquiteturais. Uma vez que cada estilo arquitetural foca em diferentes aspectos da arquitetura é possível afirmar que esse foco específico facilita atingir determinados atributos de qualidade, conforme relacionado no quadro 5.2.

Tipo de Visão	Estilo Arquitetural	Utilidade	Atributos de qualidade que podem ser beneficiados
Módulos	Decomposição	Alocação de recursos, Planejamento e estruturação de projetos, proteção da informação, encapsulamento, controle de configuração	Manutebilidade
	Uso	Extensão de Funcionalidade, Extração de subconjunto de funcionalidade	Testabilidade
	Generalização	Incremento de funcionalidade, reuso, detecção de comportamentos similares	Manutebilidade e Reuso
	Camada	Desenvolvimento incremental, implementação de máquina virtual	Manutebilidade e Portabilidade
Componentes e Conectores	Canos e Filtros	Sistemas que requerem vários estágios de processamento	Performance
	Dados compartilhados	Análise de performance, integridade de dados e modificabilidade	Performance, Segurança, Confiabilidade
	Publicador-Subscritor	Envio de mensagens para receptores desconhecidos	Manutebilidade
	Cliente Servidor	Operação distribuída, separação de responsabilidade, análise de performance, balanceamento de carga	Performance, Escalabilidade, Confiabilidade
	Peer-to-Peer	Processamento distribuído, processamento intensivo	Performance
	Processos Comunicantes	Análise de processos concorrentes e de performance	Performance e Confiabilidade
Alocação	Distribuição	Análise de performance, disponibilidade e integridade de dados	Performance, Reuso, Segurança
	Implementação	Atividades de controle de configuração, integração e testes	Manutebilidade e Testabilidade
	Atribuição de Trabalho	Gerenciamento de projetos, melhor uso de especialistas, gerenciamento de recursos	Estimativa de Tempo e Custo, Alocação de recursos,

Quadro 5.2 - Relacionamento entre estilos arquiteturais e atributos de qualidade

5.5. Conclusões do Capítulo

A manutenção e a evolução de um produto de software é uma conseqüência natural da evolução dos negócios e ambiente operacional o que faz com que o software fique cada vez maior e mais complexo. Para ajudar a lidar com esse crescimento e aumento de complexidade, a arquitetura de software fornece um nível de abstração que facilita o tratamento de questões relacionadas a manutebilidade, tolerância a falhas, performance, acurácia, interoperabilidade e portabilidade. A arquitetura pode: servir como base de conhecimento; proporcionar o correto crescimento e evolução; facilitar a integração com outros sistemas; ser utilizada para aperfeiçoar o sistema e facilitar a correção de erros.

CAPÍTULO 6

O PROCESSO DE MIGRAÇÃO DE SO*FTWARE*

Objetivo do Capítulo

O objetivo deste capítulo é apresentar um processo de migração gradual de sistema legado funcional para orientado a objetos e direcionado pelos indicadores de qualidade, demonstrando como ele é organizado, quais são seus sub-processos e atividades, além de destacar as entradas, artefatos gerados, ferramentas ou mecanismos e responsáveis por cada atividade.

A utilização dos indicadores de qualidade tem como meta ajudar a definir a arquitetura mais adequada e facilitar o atendimento dos requisitos do sistema.

A descrição do processo apresentado neste capítulo segue uma estrutura onde podem ser utilizados os seguintes itens:

- Considerações gerais: onde são descritos os objetivos, restrições e detalhes de cada etapa ou atividade, sempre que necessário;
- Diagramas: IDEF0 (*Integration Definition for Function Modeling*) para mostrar graficamente as atividades de cada processo;
- Quadros de apoio ao diagrama IDEF0: sumarizando as atividades e relacionando seus insumos ou artefatos de entrada (E), produtos ou artefatos de saída (S) que podem ser insumos de atividades seguintes, Mecanismos (M) onde podem ser citadas metodologias, ferramentas ou quaisquer técnicas de

auxílio à solução do problema, e principais envolvidos e responsáveis na execução da atividade em questão (R).

Para algumas atividades também serão apresentados alguns instrumentos de apoio, os quais auxiliam, apresentam alguma técnica ou orientam a maneira adequada para o desenvolvimento da atividade em questão.

6.1. O Processo de Migração de Sistemas Legados para OO

A proposição desse processo visa resolver o problema de migrar sistemas legados funcionais desenvolvidos em COBOL, na plataforma AS/400 ou *Mainframe*, caros de manter, ultrapassados quanto a metodologias de desenvolvimento de software e limitados quanto ao aproveitamento de tecnologias mais recentes, tudo isso de forma a permitir como resultado a geração um sistema que possa ter uma qualidade superior ao atual.

Para resolver o problema de migração de sistemas legados para atender requisitos funcionais e não funcionais, os últimos não são considerados nos processos de migração voltados para o código fonte (vide item 3.1.3), propomos o processo a ser descrito nos próximos tópicos.

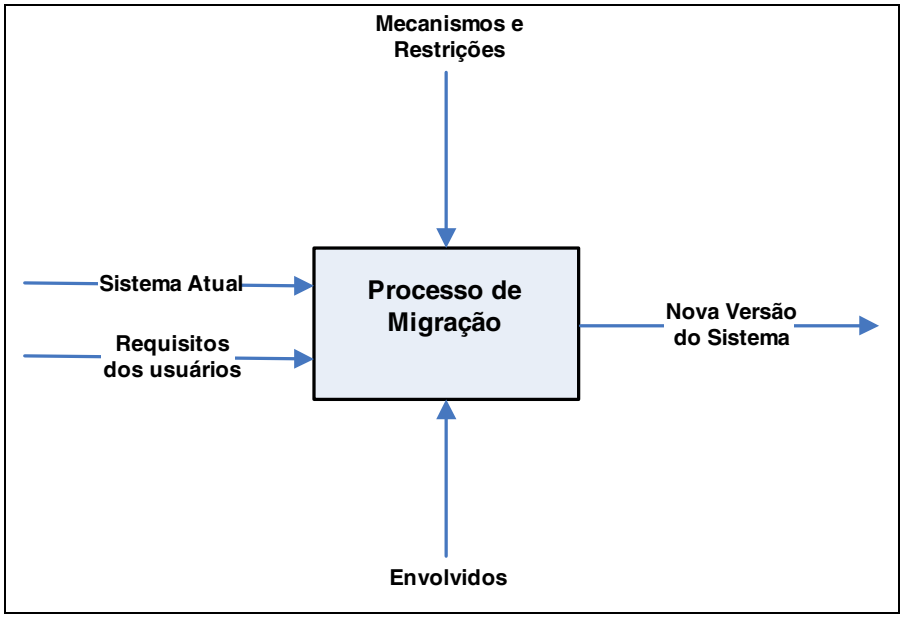

Figura 6.1 – Visão Geral do Processo

A figura 6.1 apresenta a visão geral do processo, onde:

- Sistema atual: é o sistema de software onde será aplicado o processo. O sistema deve estar funcionando em ambiente de produção e ter funções efetivas de entrega aos usuários. Quanto maior a demanda por inovações para atender a novos requisitos, maior será a justificativa para aplicar o processo de transformação;
- Requisições dos usuários: são todas as solicitações de melhoria feitas pelos usuários, incluindo, por exemplo, reclamações de baixa performance, dificuldade de operação, demora na entrega de novas solicitações e quaisquer outras demandas que devam ser atendidas pela equipe de manutenção;
- Nova versão do sistema: corresponde a um novo aplicativo migrado para o paradigma OO que considerou os requisitos funcionais e não funcionais nas metas de qualidade a serem atingidos;

- Mecanismos e restrições: são regras e limitações do ambiente de software e hardware sobre os quais o sistema trabalha, normas internas, metodologias e modelos;
- Envolvidos: são todas as pessoas que estejam diretamente ligadas ao sistema, seja testando, modificando, enviando ou recebendo informações. Nesse trabalho, consideramos como premissa o envolvimento dos seguintes profissionais:
    - Gerente de Projeto;
    - Analistas de Sistemas;
    - Equipe de Qualidade de Software;
    - Analista de Produção;
    - Usuários do sistema.

O escopo desse trabalho é voltado para os aplicativos desenvolvidos em ambientes AS/400 ou Mainframe, utilizando o paradigma funcional, codificados em linguagens como COBOL e baseado em arquitetura de duas camadas. Este tipo de sistema é caracterizado por ser desenvolvido para instituições financeiras, usando técnicas e representações da Análise Estruturada (DFD, MER), com acesso direto à base de dados. É assumido também o acesso ao código fonte, bem como ao esquema de base de dados e de arquivos.

O processo é uma variante do modelo evolucionário incremental do ciclo de vida de software, e tem como ponto de partida o sistema atual, sob o qual serão analisadas as condições de funcionamento e os resultados que ele apresenta (indicadores operacionais). Como saída tem-se uma nova versão do sistema implementado sob o paradigma da orientação a objetos.

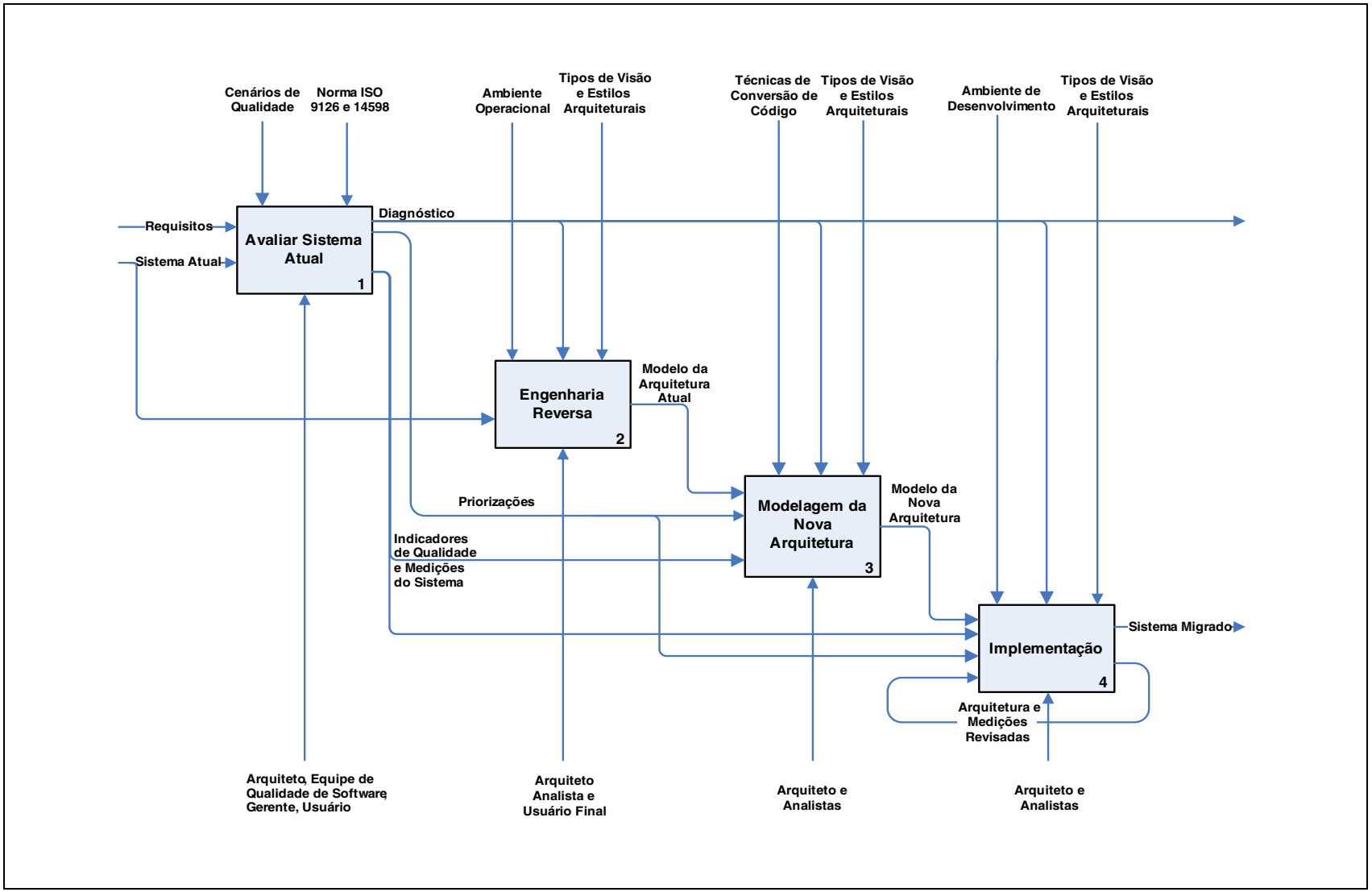

Figura 6.2 - Processo A0: Transformação de Arquitetura

A proposta desse trabalho é que, a partir da definição da arquitetura OO, o processo seja repetido no sub-processo de implementação, onde a qualidade do sistema será elevada a patamares mais altos (figura 6.2).

As etapas previstas neste processo são:

- Avaliar Sistema Atual: nesta etapa será realizada uma avaliação do tipo caixa preta considerando as visões do usuário, desenvolvedor e gerente de sistemas;
- Engenharia Reversa: será abstraído o modelo arquitetural do sistema legado, utilizando os tipos de visão e estilos arquiteturais como forma de ter uma visão mais precisa do sistema;
- Modelagem da Nova Arquitetura: um modelo de arquitetura OO de todo o sistema será obtido nesta etapa;
- Implementação: esta etapa incremental consistirá da realização do desenvolvimento, implementação e testes do sistema e uma das suas saídas será utilizada para realimentar os indicadores de qualidade e a própria arquitetura nas iterações seguintes.

No quadro 6.1 temos um resumo das atividades do processo.

Atividades	Elementos
Avaliar sistema atual É a atividade onde será uma avaliação do tipo caixa preta.	E: Requisitos Sistema Atual
	S: Diagnóstico (migrar sistema ou não) Indicadores de Qualidade Priorizações Medições do Sistema atual
	M: Norma ISO 9125 Norma ISO 14598 Cenários de Qualidade
	R: Arquiteto Equipe de Qualidade de Software Usuários Gerente de Projeto
Engenharia Reversa Gerar modelo arquitetural do sistema legado, utilizando as visões e estilos arquiteturais.	E: Diagnóstico Sistema Atual
	S: Modelo da Arquitetura Atual
	M: UML DER Arquitetura de Software
	R: Arquiteto
Modelagem da Nova Arquitetura Gerar modelo da nova arquitetura sob o paradigma da orientação a objetos.	E: Modelo da Arquitetura Atual Indicadores de Qualidade Medições do Sistema Atual Priorizações
	S: Modelo da Nova Arquitetura
	M: UML Arquitetura de Software
	R: Arquiteto Desenvolvedor
Implementação É o desenvolvimento e a implementação gradual do novo sistema sob o paradigma da orientação a objetos, envolvendo software, hardware, documentos e processos.	E: Modelo da Nova Arquitetura Indicadores de Qualidade Medições do Sistema Atual Priorizações
	S: Sistema Migrado Parcialmente
	M: UML Arquitetura de Software
	R: Arquiteto Desenvolvedor
Legenda: E: insumos de entrada S: produtos de saída M: mecanismos, técnicas ou ferramentas utilizadas, regras e leis R: responsáveis pela execução da atividade	

Quadro 6.1 – Sub-processos do "Processo de Migração do Sistema Legado"

As etapas do processo proposto tem origem nas pesquisas realizadas por De Lucia (1997), Bodhuin (2002) e Ying (2004), que contemplam a migração de sistemas legados para orientado a objetos, e na proposta de desenvolvimento de arquitetura para atingir requisitos não funcionais proposto por Bachmann (2005).

As etapas que compõem o processo foram criadas de forma a atingir melhor os seguintes requisitos:

- Reduzir o alto custo de manutenção do sistema legado procedural;
- Reduzir o prazo das manutenções cada vez maior;
- Permitir a utilização de novas linguagens e metodologias de desenvolvimento;
- Melhorar a qualidade do sistema;
- Evitar os riscos de criação de um novo sistema para substituir o antigo.

6.2 Avaliar Sistema Atual

Para Ying (2003), a qualidade de software é definida através de um conjunto de características do produto de software relacionados a atributos externos, como performance, e a atributos internos como complexidade e estrutura de dados. Os atributos externos se referem principalmente ao ambiente operacional onde o software é executado. Os atributos internos estão relacionados a características do código fonte que podem ser mensuradas através de métricas apropriadas. Ambos os tipos de atributos são relevantes e podem ser interdependentes, onde, por exemplo, temos o atributo externo manutebilidade dependendo de atributos internos como alta coesão e baixo acoplamento.

A avaliação do sistema atual tem como objetivo verificar sua qualidade, através de métricas baseadas nas características e sub-características propostas pela norma NBR ISO/IEC 9126, identificar seus principais problemas e gerar um diagnóstico.

A identificação desses problemas ajudará a definir se a migração deverá ser realizada ou não (diagnóstico). É importante associar o custo a cada problema identificado de forma a poder medir em termos financeiros o quão válido será entrar nesta empreitada, mas esse não é o escopo desse trabalho. Nesta discussão devem-se considerar as informações como indicadores de desempenho, necessidades não atendidas, reclamações compiladas, listas de pendências e histórico de erros. A figura 6.3 ilustra os processos envolvidos nesta etapa que serão detalhados nos próximos tópicos.

Este processo foi criado desta forma para poder atingir os seguintes requisitos:

- Avaliar o aplicativo sem entrar no mérito do seu funcionamento;
- Considerar ponto de vista de quem usa e paga as solicitações de alteração no sistema;
- Gerar diagnóstico incluindo decisão se o sistema será migrado ou não, considerando as priorizações.

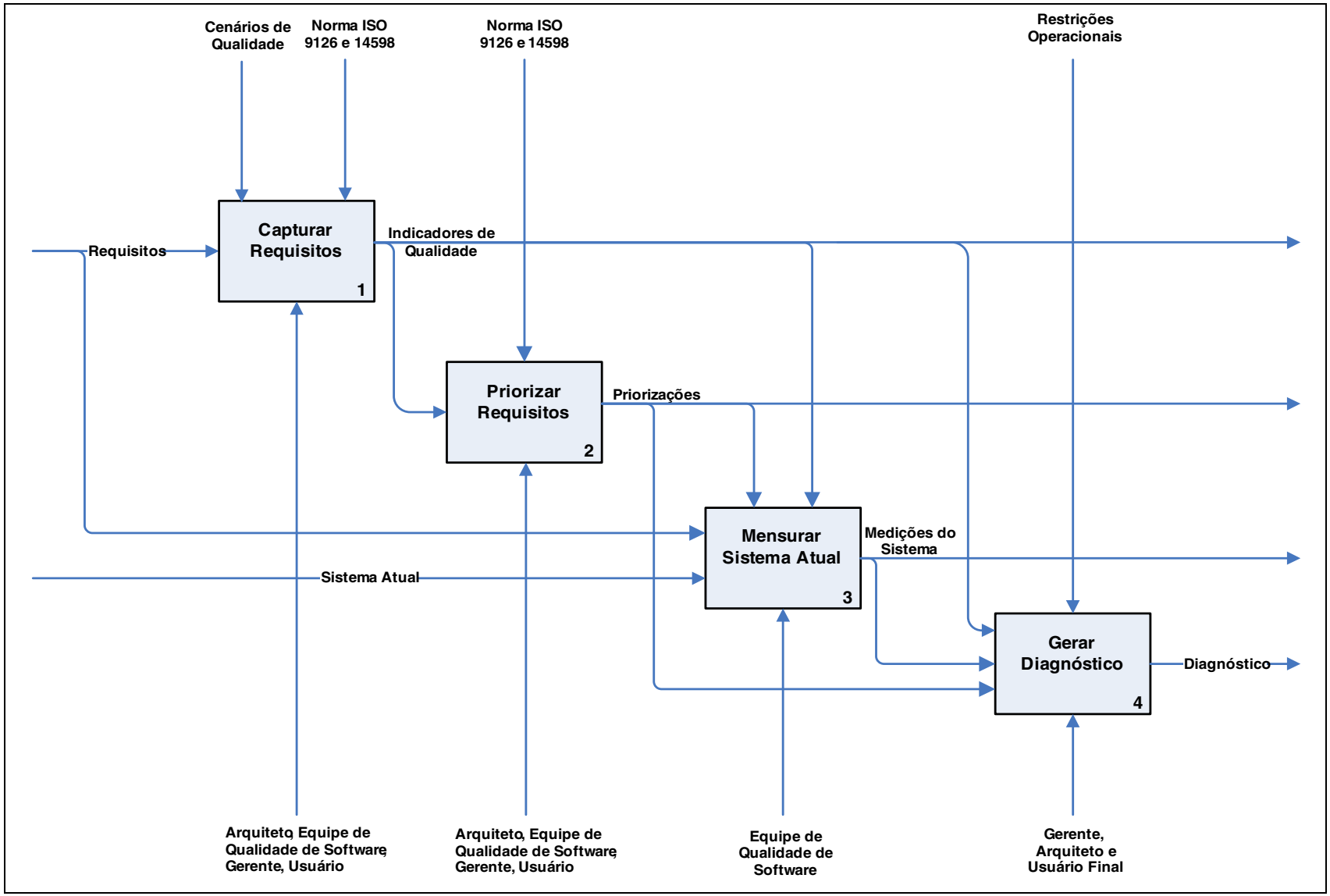

Figura 6.3 - Processo A1: Avaliação do Sistema Atual

6.2.1 Capturar Requisitos

Através dessa etapa (figura 6.4) serão capturados os requisitos necessários para a avaliação do sistema. Essa captura se dará através de entrevistas com os usuários, utilização de questionários e de documentos que evidenciem problemas e soluções relacionados ao dia-a-dia do sistema (histórico de ocorrências), pendências, desempenho do sistema e da equipe de desenvolvimento quanto a atividades de manutenção e evolução do sistema.

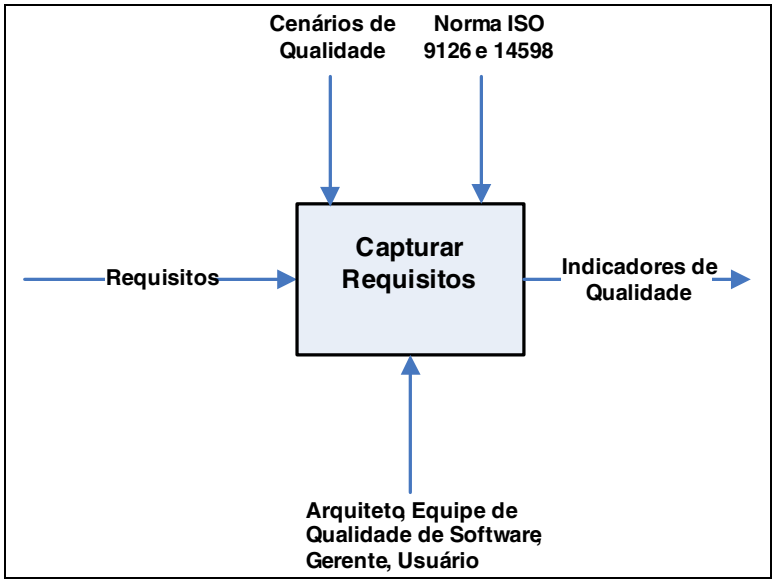

Figura 6.4 – Sub-processo Capturar Requisitos

Como ferramentas de apoio à captura de requisitos e definição de metas a serem atingidas pelo sistema, utilizaremos as normas ISO 9126 e 14598 e os cenários de qualidade proposto por Bass et al. (2003) que são constituídos das seguintes partes:

- Fonte de estímulo: é uma entidade (pessoa, sistema ou outro ator) que gera o estímulo;
- Estímulo: é um evento que necessita ser tratado sempre que chegar ao sistema;

- Ambiente: é onde o estimulo ocorre e as condições necessárias para que ele possa ocorrer;
- Artefato: é o elemento afetado pelo estimulo. Pode ser um sistema inteiro ou partes dele, por exemplo;
- Resposta: é a atividade resultante da chegada do estímulo
- Medição da resposta: quando a resposta ocorre, ela deve ser medida de alguma forma para que possa ser testada.

Os cenários de qualidade podem ser genéricos, aplicáveis a qualquer sistema, ou especifico para o sistema.

Esses cenários genéricos estão associados às características da norma ISO 9126 e cada um deles pode ser utilizado para gerar um ou mais cenários a ser aplicado em um sistema específico. Um cenário utilizado em sistema específico está para a especificação de requisitos de qualidade assim como um cenário de caso de uso está para a especificação de requisitos funcionais.

No quadro 6.2 é apresentada uma relação de cenários genéricos de qualidade que podem ser utilizados como base para geração dos cenários específicos.

Característica de qualidade	Fonte do estímulo	Estimulo	Ambiente	Artefato	Resposta	Medição da Resposta
Disponibilidade	Interna ou externa ao sistema	Omissão de um componente ao emitir uma resposta (repetitiva ou não); Resposta gerada pelo componente muito cedo ou tarde demais; Resposta de um componente gerada de forma incorreta	Sistema Operando de forma normal ou sobrecarregado	Processadores; Canais de comunicação; Memória	Registro de falhas; Notificação dos envolvidos; Ficar indisponível em intervalos pré-determinados; Continuar operando de forma normal ou degradada	Intervalo de tempo que o sistema deve ficar disponível; Tempo que o sistema fica disponível; Intervalo de tempo que o sistema pode operar de forma degradada; Tempo de recuperação do sistema
Modificabilidade	Usuário final; Desenvolvedor; Adminstrador do sistema	Alteração, criação ou eliminação de funcionalidade, atributo de qualidade ou capacidade do sistema	Desenvolvimento; Homologação; Implantação	Interface com usuário; Plataforma de desenvolvimento; Outros sistemas	Localização de lugares a serem modificados; Realização da modificação sem afetar outras funcionalidades (ou não); Modificação dos testes; implementaçao da modificação	Número de elementos afetados direta ou indiretamente pela modificação; Esforço necessário; custo gerado;

Quadro 6.2 – Cenários de Atributo de Qualidade

Característica de qualidade	Fonte do estímulo	Estimulo	Ambiente	Artefato	Resposta	Medição da Resposta
Eficiência	Um conjunto de usuários externos ou internos	Execução de várias transações por unidade de tempo; Chegada de eventos periódicos, esporádicos e estocásticos	Sistema Operando de forma normal ou sobrecarregado	Sistema	Processamento da resposta ao estimulo; Mudança do nível de serviço	Tempo para processamento de eventos e a variação deste tempo; Quantidade de eventos processados em um particular intervalo de tempo; Identificação de eventos com características que os impedem de ser processados

Quadro 6.2 – Cenários de Atributo de Qualidade (continuação)

Característica de qualidade	Fonte do estímulo	Estimulo	Ambiente	Artefato	Resposta	Medição da Resposta
Segurança	Usuários externos ou internos com identificação correta ou incorreta, com autorização ou não a certas funcionalidades, com limitação ou não de acesso;	Tentativa de acessar, modificar ou eliminar alguma informação ou serviços providos pelo sistema;	On-line ou off-line;Conectado ou desconectado;Protegido por Firewall ou não	Serviços providos pelo sistema; Dados dentro do sistema	Autenticação do usuário; Proteção das informações do usuário; Bloqueio ou permissão de acesso a serviços e dados; Registros de acesso ou modificações (e tentativas); Reconhecimento (e devido informe) de inesperadas altas demandas a determinados serviços;	Tempo, recurso e esforço necessário para prever medidas de segurança com probabilidade de sucesso; Probabilidade de detenção de ataques; Probabilidade de identificar o responsável por acesso, modificaçao de dados ou serviços e ataques; Percentual de serviços que permanecem disponíveis após ataque por negativa de serviço

Quadro 6.2 – Cenários de Atributo de Qualidade (continuação)

Característica de qualidade	Fonte do estímulo	Estimulo	Ambiente	Artefato	Resposta	Medição da Resposta
Testabilidade	Testador de teste unitário; Desenvolvedor de unidade de software; Outros sistemas; Sistemas verificadores; Testador do cliente	Teste unitário; Teste integrado; Integração entre partes do sistema; Atividades de análise, arquitetura e projeto.	Desenvolvimento; Homologação; Implantação	Sistema inteiro; Partes do sistema; Partes do código	Permitir controlar a execução de testes desejados e observar os resultados gerados	Percentual de instruções de código executadas; Percentual de tolerância a falhas; Tempo para realização de testes; Maior nível de profundidade dos componentes envolvidos no teste Tempo de preparação do ambiente de teste;
Usabilidade	Usuário Final	Interação com usuários com diferentes níveis de aprendizado e eficiência	Sistema em execução	Sistema	O sistema responde de forma adequada ou não com as seguintes características: facilidade de entendimento de seu funcionamento; eficiência no seu uso; minimização no impacto de erros; adaptação; nível de conforto	Tempo de resposta; Número de erros; Número de problemas solucionados; Nível de satisfação do usuário; ganho de conhecimento pelo usuário; taxa de operações realizadas com sucesso; dados perdidos por unidade de tempo

Quadro 6.2 – Cenários de Atributo de Qualidade (continuação)

Segundo Cockburn (2000), é necessário que os requisitos de qualidade sejam mensuráveis. Como exemplo de especificação correta, o autor cita que não se deve dizer “a manutenção deste sistema é cara” e sim “a manutenção deste sistema custa R$100/hora”.

Para Ying (2003), a qualidade do software também pode estar relacionada aos objetivos do software, porque o grau de conformidade com requisitos não funcionais pode ser definido em termos relativos ao invés de absolutos.

A aplicação de cenários genéricos em um sistema especifico, sugerida por Bass et al. (2003) auxilia a definir requisitos de qualidade em termos operacionais. No quadro 6.3 é apresentado um exemplo de cenário específico, onde na coluna “medição da resposta” sugerimos a utilização de indicadores de qualidade (métrica) com valores desejáveis para o sistema a ser migrado e que serão usados na medição do sistema atual.

Cenários	Fonte do estímulo	Estimulo	Ambiente	Artefato	Resposta	Medição da Resposta
Cenário1 (disponibilidade)	Externa ao sistema	Chegada de mensagem não prevista	De produção operando de forma normal	Processo de captura de mensagens	Informa ao operador da situação ocorrida	Sem queda do sistema
Cenário2 (modificabilidade)	Desenvolvedor	Solicitação de mudança da interface com o usuário	Desenvolvimento	Funcionalidade do Sistema	Sistema opera com a nova modificação	3 horas x homem
Cenário3 (eficiência)	Usuário	Início de uma transação	De produção operando de forma normal	Funcionalidade do Sistema	Transação é processada normalmente	Tempo de resposta médio de 2 segundos
Cenário4 (segurança)	Usuário identificado corretamente	Tentativa de restaurar uma modificação gerada de forma indevida	De produção operando de forma normal	Dados dentro do sistema	Sistema verifica trilha de auditoria	Dados corretos restaurados em 1 dia
Cenário5 (testabilidade)	Equipe de Qualidade de Software	Realização de teste unitário	Desenvolvimento	Componente do sistema	Apresenta interface que permite observar o controle e saídas geradas	85% dos testes realizados em menos de 3 horas
Cenário6 (Usabilidade)	Usuário	Cancelamento de uma operação gerada indevidamente	De produção operando de forma normal	Sistema	Operação é cancelada	Operação é cancelada em menos de 1 segundo
Cenário7 (Usabilidade)	Usuário	Consulta uma operação gerada há um mês	De produção operando de forma normal	Sistema	Sistema apresenta interface com resultado da solicitação	Operação é realizada em tempo médio de 2 segundos

Quadro 6.3 – Cenários aplicados a um sistema específico

6.2.2 Priorizar Requisitos

Segundo Cockburn (2000), os requisitos de qualidade devem ser priorizados e não se deve considerar que todos são importantes.

A decisão do que é crítico ou não deve ser tomada pelo gerente e pelos usuários do sistema, com apoio da equipe de qualidade de software, que fornecerão os elementos para tomada de decisão.

Esta etapa também gera como artefato uma lista de prioridades, elaborada a partir dos próprios requisitos capturados na etapa anterior. O quadro 6.4 apresenta um exemplo de sugestão de lista de prioridades.

Priorização	Requisito	Métrica	Valores Desejados
1	Tratamento de mensagens não previstas (tolerância a falhas)	Falhas evitadas/total de testes	90%
2	Modificação de funcionalidade	Erros pós-implantação/ total de implantações	5%
3	Diminuir tempo de desenvolvimento a solicitações de mudança	Horas	Redução de 30%
4	Diminuir custo de desenvolvimento a solicitações de mudança	Horas x Homem	Redução de 20%

Quadro 6.4 – Exemplo de priorizações

6.2.3 Mensurar Sistema Atual

Uma vez definidos os indicadores de qualidade (cenários) e estabelecidas as priorizações obtidas na etapa anterior, deve-se proceder com as medições que serão utilizadas para elaboração do diagnóstico do sistema atual.

A medição deve ser realizada pela equipe de qualidade de software através de interação com o sistema e da análise dos documentos capturados. Os cenários serão utilizados como ferramenta de apoio e a saída será um documento com o resultado das medições realizadas. O quadro 6.5 sugere um exemplo a ser utilizado como modelo para registro do resultado dessa atividade. A diferença desse quadro para o quadro 6.3 é que a coluna medição da resposta irá representar os valores efetivamente medidos.

Cenários	Fonte do estímulo	Estimulo	Ambiente	Artefato	Resposta	Medição da Resposta
Cenário1 (disponibilidade)	Externa ao sistema	Chegada de mensagem não prevista	De produção operando de forma normal	Processo de captura de mensagens	Sistema cancela	100% de queda de sistema
Cenário2 (modificabilidade)	Desenvolvedor	Solicitação de mudança da interface com o usuário	Desenvolvimento	Funcionalidade do Sistema	Sistema opera com a nova modificação	8 horas x homem
Cenário3 (eficiência)	Usuário	Início de uma transação	De produção operando de forma normal	Funcionalidade do Sistema	Transação é processada normalmente	Tempo de resposta médio de 5 segundos
Cenário4 (segurança)	Usuário identificado corretamente	Tentativa de restaurar uma modificação gerada de forma indevida	De produção operando de forma normal	Dados dentro do sistema	Sistema não tem trilha de auditoria para todas as situações	Dados corretos restaurados de forma parcial em 1 dia
Cenário5 (testabilidade)	Equipe de Qualidade de Software	Realização de teste unitário	Desenvolvimento	Componente do sistema	Não há interface que permita observar o controle e saídas geradas	55% dos testes realizados em 24 horas
Cenário6 (Usabilidade)	Usuário	Cancelamento de uma operação gerada indevidamente	De produção operando de forma normal	Sistema	Operação é cancelada de forma manual	Operação é cancelada em menos de 1 dia
Cenário7 (Usabilidade)	Usuário	Consulta uma operação gerada há um mês	De produção operando de forma normal	Sistema	Sistema apresenta interface com resultado da solicitação	Operação é realizada em tempo médio de 10 segundos

Quadro 6.5 – Cenários aplicados para registrar o resultado das medições

6.2.4 Gerar Diagnóstico

Após a definição do nível de qualidade do sistema atual e dos níveis desejados, ocorre o processo onde é decidido se o sistema atual deverá ser migrado ou não. Deverão ser considerados aspectos operacionais como:

a) Disponibilidade e custo de programadores com habilidade para atuar na plataforma atual;

b) Disponibilidade e custo dos programadores disponíveis para atuar em uma das plataformas para a qual o sistema será migrado;

c) Indicadores de eficiência da linguagem atual versus indicadores de eficiência da futura linguagem;

d) Custo de manutenção do sistema atual versus custo de desenvolvimento do sistema em uma nova arquitetura e paradigma;

e) Custo da infra-estrutura que suporta o sistema atual versus custo da nova infra-estrutura;

Vale ressaltar que, no caso de decisão pela migração, é importante que os analistas que estiverem mais envolvidos com o processo de negócio sejam treinados para o desenvolvimento sob novo paradigma.

O resultado deste processo será um documento que contenha:

- Aspectos analisados na etapa de avaliação do sistema
- Referência aos documentos colhidos, entrevistas e medições realizadas;
- Identificação dos principais problemas do sistema;

- Motivos que justifiquem a necessidade de migração ou não;
- Vantagens a serem obtidas com a migração;
- Metas a serem atingidas com o novo sistema.

6.3 Engenharia Reversa

As atividades relativas à engenharia reversa tem por objetivo recuperar o estado atual da arquitetura e da implementação do aplicativo. É nessa etapa onde as informações do sistema são extraídas através de documentação, código fonte, entrevistas com os usuários e desenvolvedores. Esse conjunto de etapas vai proporcionar o entendimento, análise e discussão sobre características do sistema. A idéia é obter o detalhamento de aspectos técnicos e de negócio já implementados.

Segundo Magalhães e Arakaki (2005), para se obter um entendimento global do sistema há a necessidade de se ter, no mínimo, uma representação arquitetural para cada grupo de interesse. Exemplo: IDEF0 para os processos de negócio, endereçado aos usuários do sistema; diagramas de classe, diagramas de casos de uso e diagramas de entidade e relacionamento para apoiar o entendimento dos desenvolvedores de software e diagramas de distribuição para entender como o sistema está distribuído através do hardware.

Para realizar o mapeamento das necessidades de documentos arquiteturais, Clements et al. (2003) sugerem a construção de uma tabela que determine para cada envolvido o tipo de visão e o estilo arquitetural necessário. No quadro 6.6 apresentamos um exemplo do nível de detalhamento de cada estilo arquitetural a ser visualizado pelos envolvidos.

	Tipos de Visão				
	Módulos		Componentes e Conectores	Alocação	
Envolvido	Decomposição	Uso	Canos e Filtros	Distribuição	Atribuição de Trabalho
Gerente de projeto	D	-	G	G	D
Analista de sistemas	D	-	D	-	-
Equipe de Qualidade	D	G	D	-	G
Analista de Produção	G	G	G	D	-
Usuários do Sistema	G	-	G	G	-
Legenda: D - informação detalhada; M – grau médio de detalhamento; G – informação geral					

Quadro 6.6 - Estilos arquiteturais sugeridos (Clements et al., 2003, p330-331).

A proposta é que este sub-processo (ilustrado na figura 6.4) seja realizado através da execução de 4 atividades:

- Captura do processo de negócio;
- Análise dos dados;
- Análise dos procedimentos;
- Extração do modelo arquitetural.

As atividades “Capturar processos de negócio”, “Modelar dados” e “Análise dos procedimentos” estão incluídas no contexto de se utilizar representações de estilos arquiteturais como forma de capturar melhor o entendimento do sistema atual e principalmente para apoiar o mudança de paradigma, que necessita de um foco especial nessas etapas.

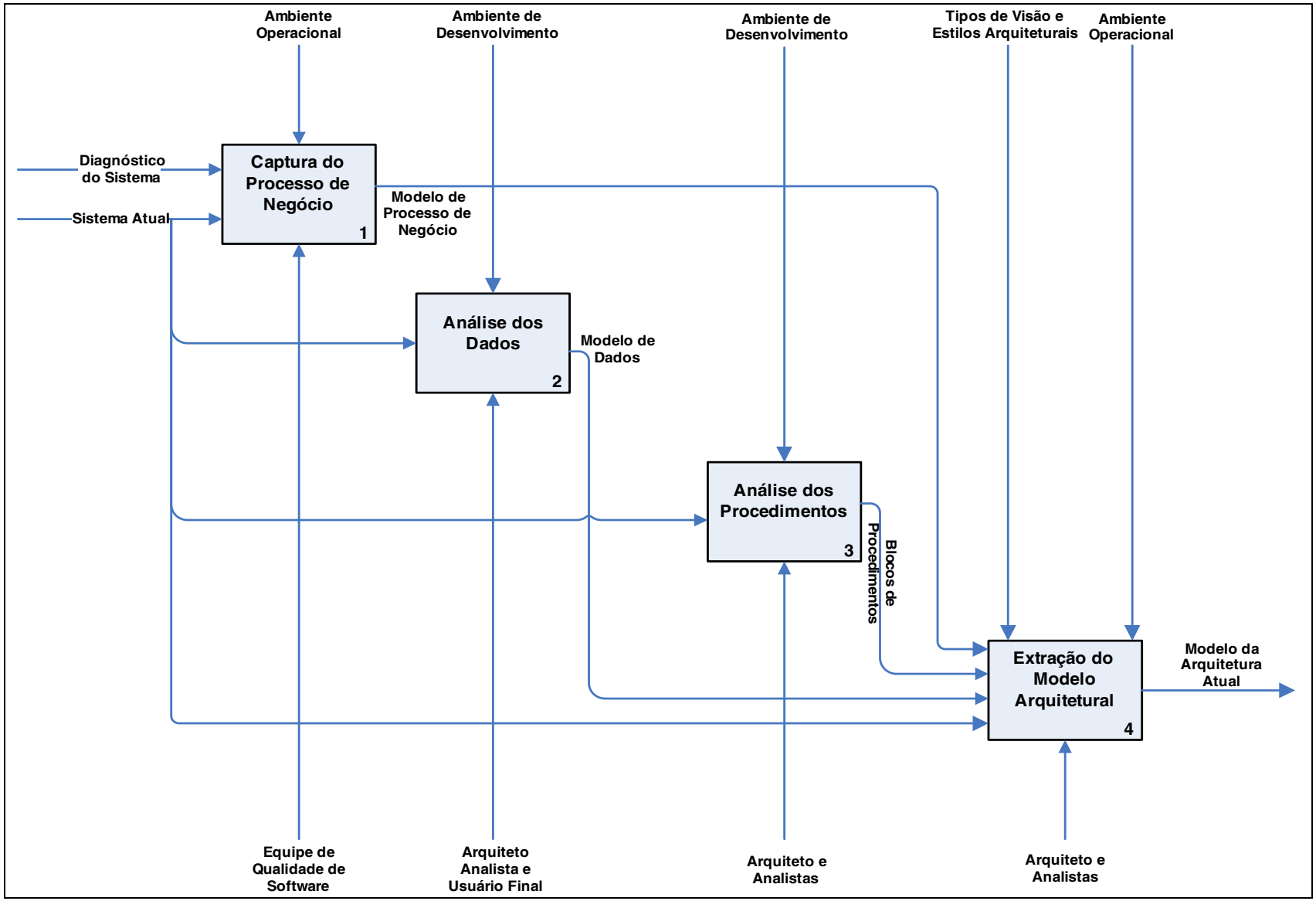

Figura 6.4 - Processo A2: Engenharia Reversa

Como resultado desse conjunto de atividades, teremos o modelo do sistema atual compostos por todos os artefatos gerados (descrito nos próximos tópicos).

Os seguintes requisitos a elaboração deste processo e respectivas etapas:

- Obter um entendimento do sistema atual considerando plataforma, ambiente de desenvolvimento, estrutura da implementação, entre outros;
- Facilitar a segmentação do sistema, priorização e a identificação de requisitos de alto nível (Captura de processos de negócio);
- Identificar elementos (estrutura de dados e blocos de procedimentos) que serão candidatos a classes e métodos na nova arquitetura OO (Análise dos dados e procedimentos);
- Identificar características do sistema que o leva a ter determinados atributos de qualidade.

6.3.1 Captura do Processo de Negócio

Para o entendimento correto de um sistema, é necessário conhecer o negócio que ele apóia. Esse processo permitirá o gerenciamento da migração gradual do sistema, através da segmentação e priorização dos processos a serem migrados. Esta etapa não mantém vínculo com a tecnologia, o que a torna independente de linguagens e metodologias, e ainda permite uma visibilidade tanto em alto nível como em baixo nível dos processos internos, podendo existir uma clara visão do que poderá ser migrado primeiro. Na figura 6.5 apresentamos um exemplo de representação em IDEF0 de processo capturado.

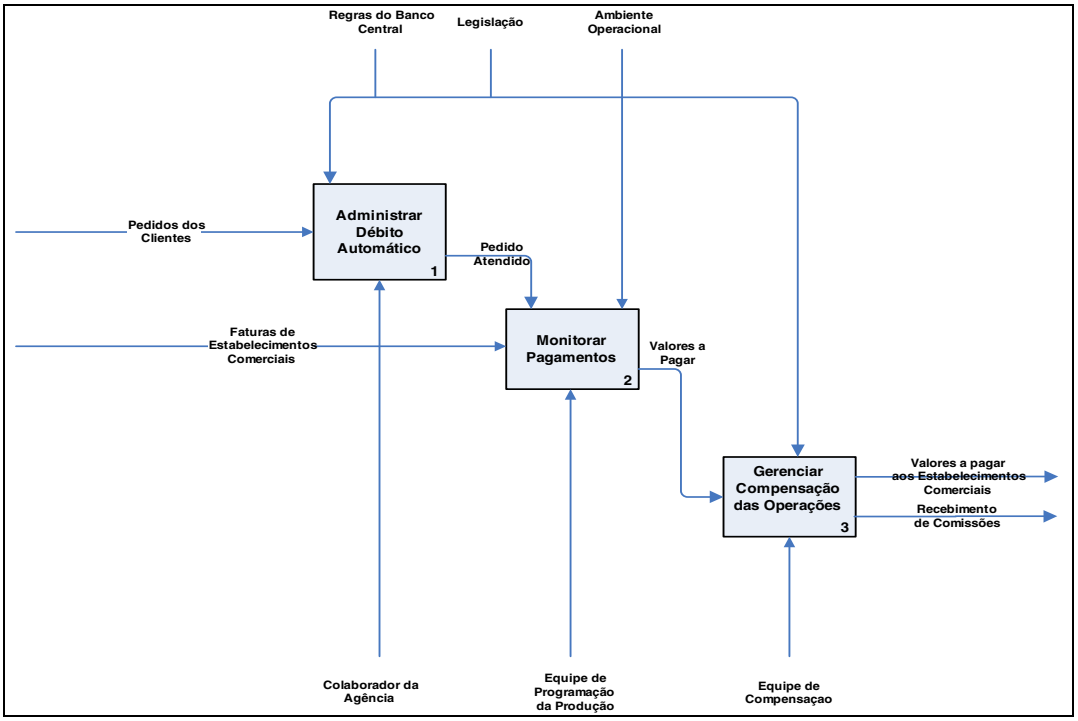

Figura 6.5 - Exemplo de representação de processo capturado

6.3.2 Análise dos Dados

Para se conhecer a estrutura do sistema, este trabalho assume que as informações relativas a código fonte, base de dados, estrutura de arquivos e demais objetos está disponível.

Nessa atividade, o código fonte é analisado de forma a recuperar toda informação necessária para a realização das atividades seguintes. Em particular, esta tarefa visa identificar nos arquivos do sistema aqueles objetos que modelem conceitos fundamentais do domínio de aplicação do sistema legado. Este conjunto de objetos irá ajudar na elaboração da arquitetura do sistema que envolve a análise das seguintes informações:

a) Procedimentos existentes;

b) Estrutura de dados (variáveis locais e globais);

c) Arquivos;

d) Banco de dados;

e) Chamadas de subprogramas;

f) Interfaces com outros programas;

g) Interfaces com o usuário (telas, páginas).

Programas legados de aplicações bancárias muitas vezes são escritos, por exemplo, em COBOL, acessam banco de dados e arquivos e possuem uma estrutura de dados internas. Estas estruturas de dados, juntamente com os bancos de dados e arquivos formam um grupo de elementos que são geralmente denominados de objetos candidatos.

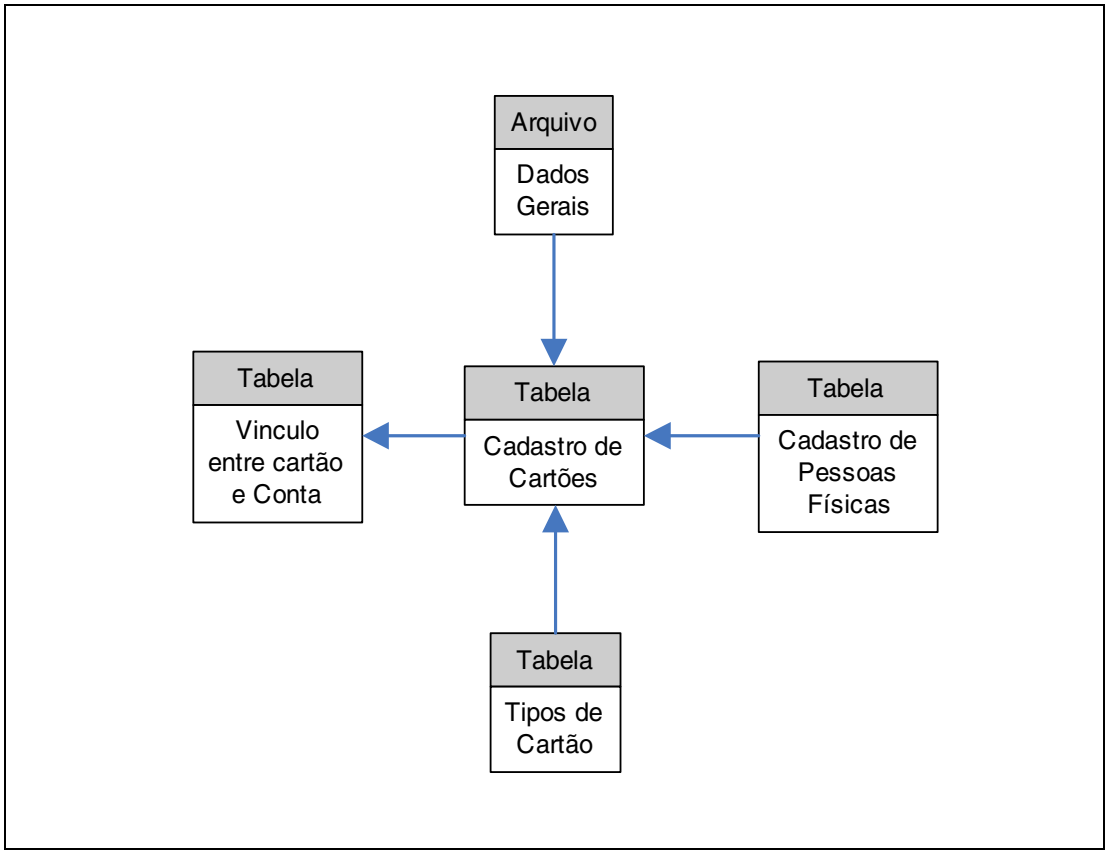

Figura 6.6 – Exemplo de Diagrama Entidade x Relacionamento

Como artefato dessa atividade, sugeri-se a utilização do diagrama Entidade-Relacionamento apresentado na figura 6.6, onde cada arquivo e tabela representam uma

estrutura de dados (não detalhada no exemplo) e é um potencial candidato a classe no novo sistema.

6.3.3 Análise dos Procedimentos

Programas codificados em linguagens como COBOL e RPG podem ser classificados como Batch ou On-line. Programas On-line possuem embutidos em si componentes que gerenciam a interface com o usuário juntamente com componentes de domínio da aplicação. Programas batch podem possuir componentes de gerenciamento de interface com outros programas e componentes de domínio de aplicação.

Componentes de interface com o usuário incluem procedimentos que comandam direta ou indiretamente o controle das instruções de entrada e saída. Desta forma, em um sistema cliente servidor, o cliente implementará a interface com o usuário, controlará a interação e requisitará os serviços necessários para o servidor.

Através da análise do código fonte, é possível obter informações que complementem o modelo de dados obtido na análise dos dados.

Essa etapa consiste na análise de módulos, subprogramas, buscando informações que auxiliem o processo de reengenharia, e que proporcionem informações úteis para elucidar a arquitetura do sistema. Como resultado dessa atividade, sugerimos a utilização do quadro 6.7 apresentado a seguir.

Programa	Tipo	Procedimento/Subprograma
Programa A	On-line	Procedimento A1
		Subprograma A1
		Procedimento A2
		Subprograma A2
		Procedimento A3
		Procedimento A4
Programa B	Bach	Procedimento B1
		Subprograma A1
		Procedimento B2
		Subprograma A2
		Procedimento B3
		Procedimento B4
Programa C	Bach	Procedimento C1
		Subprograma A1
		Procedimento C2
		Subprograma C2
		Procedimento C3
Programa D	On-line	Procedimento D1
		Subprograma A1
		Procedimento D2
		Subprograma A2
		Procedimento D3
		Procedimento D4
		Procedimento D5

Quadro 6.7 – Programas x Procedimentos

6.3.4 Extração do Modelo Arquitetural

A extração da arquitetura de um sistema é uma fase onde os elementos levantados nas etapas anteriores serão organizados e complementados com mais visões para se obter o modelo arquitetural do sistema atual.

O uso de tipos de visões e respectivos estilos arquiteturais será realizado para melhor compreender o sistema e sugerir uma nova arquitetura a ser elaborada nas próximas etapas. A utilização do modelo de processo de negócios, extraído na fase anterior, também segue a mesma linha (é mais para auxiliar a abstração do modelo arquitetural do sistema atual do que uma orientação para uma nova arquitetura).

No capítulo 5 descrevem-se os tipos de visão e respectivos estilos arquiteturais e os relacionamentos com atributos de qualidade que eles podem ajudar a atingir. Dessa forma, sugere-se a elaboração de uma planilha, semelhante ao exemplo apresentado no quadro 6.8, como forma de identificar as visões e estilos arquiteturais necessários para ajudar a atingir as metas de qualidade definidas na etapa anterior (tópico 6.2).

Etapa do processo	**Requisito**	**Artefato arquitetural sugerido**
Capturar Processos de negócio	Migrar sistema legado funcional para Orientado a Objetos	Decomposição
Analisar dados	Migrar sistema legado funcional para Orientado a Objetos	Dados compartilhados, Canos e Filtros, Distribuição, Implementação
Decompor procedimentos	Migrar sistema legado funcional para Orientado a Objetos	Generalização Decomposição Uso
Extração do Modelo arquitetural	Flexibilizar customização de perfis de acesso a funcionalidades do sistema	Distribuição, Camada, Decomposição, Cliente Servidor
	Tratamento de mensagens não previstas (tolerância a falhas)	Canos e Filtros, Processos Comunicantes, Cliente Servidor
	Modificação de funcionalidade de forma a permitir correção de erros operacionais	Dados compartilhados, Cliente Servidor, Decomposição, Uso

Quadro 6.8 – Requisitos e Artefatos Sugeridos

Etapa do processo	Requisito	Artefato arquitetural sugerido
Extração do Modelo arquitetural	Diminuir tempo de desenvolvimento a solicitações de mudança	Decomposição, Generalização, Implementação, Alocação de trabalho, Distribuição
	Diminuir custo de desenvolvimento a solicitações de mudança	Distribuição, Implementação, Alocação de Trabalho
	Melhorar tempo de resposta de determinadas transações on-line	Dados Compartilhados, Cliente Servidor, Peer-to-peer, Distribuição
	Melhorar tempos do processamento Batch	Processos Comunicantes, Canos e Filtros, Dados Compartilhados
	Diminuir quantidade de cancelamentos no Processamento Batch	Distribuição, Implementação, Processos Comunicantes, Canos e Filtros
	Modificar a interface com o usuário de forma que operar com maior eficiência	Cliente Servidor, Decomposição, Implementação
	Modificar a interface com o usuário de forma que assimilar o sistema mais rapidamente	Cliente Servidor, Decomposição, Implementação

Quadro 6.8 – Requisitos e Artefatos Sugeridos (continuação)

Com o apoio dessa planilha e com a identificação das principais características de qualidade onde o sistema atual é mais deficiente, pode-se focar melhor em aspectos que geraram esse resultado e, a partir disso, ter mais possibilidades de propor melhorias na próxima etapa. Os artefatos dessa atividade são os mesmos apresentados no quadro 6.6.

6.4 Modelagem da Nova Arquitetura

O objetivo deste sub-processo é transformar os elementos do sistema legado funcional em elementos orientados a objeto e remodelar toda a arquitetura do sistema, independendo de qual parte será transformada ou não.

Serão utilizados os artefatos gerados na etapa anterior (modelo da arquitetura atual) e, após a execução das atividades, será gerado um conjunto de artefatos, novos ou atualizados, que comporá o modelo da nova arquitetura.

De forma análoga à etapa de engenharia reversa, esse sub-processo é constituído de etapas específicas à necessidade de mudança de paradigma, a saber: eleger classes candidatas, decompor procedimentos em métodos, associar métodos a classes e de uma etapa para a modelar nova arquitetura, onde serão utilizados tipos de visões para ajudar a atender os demais requisitos de qualidade.

Assim como foi sugerido no processo Engenharia Reversa (tópico 6.3), para realizar o mapeamento das necessidades de documentos arquiteturais também utilizar-se-á uma tabela (quadro 6.6), sugerida por Clements et al. (2003), que determine para cada envolvido o tipo de visão e o estilo arquitetural necessário.

Por utilizar artefatos gerados na etapa anterior, em todas as etapas desse sub-processo poderá ocorrer o mapeamento entre os elementos arquiteturais do sistema legado e do novo sistema migrado a ser gerado. A figura 6.7 mostra as etapas envolvidas no processo.

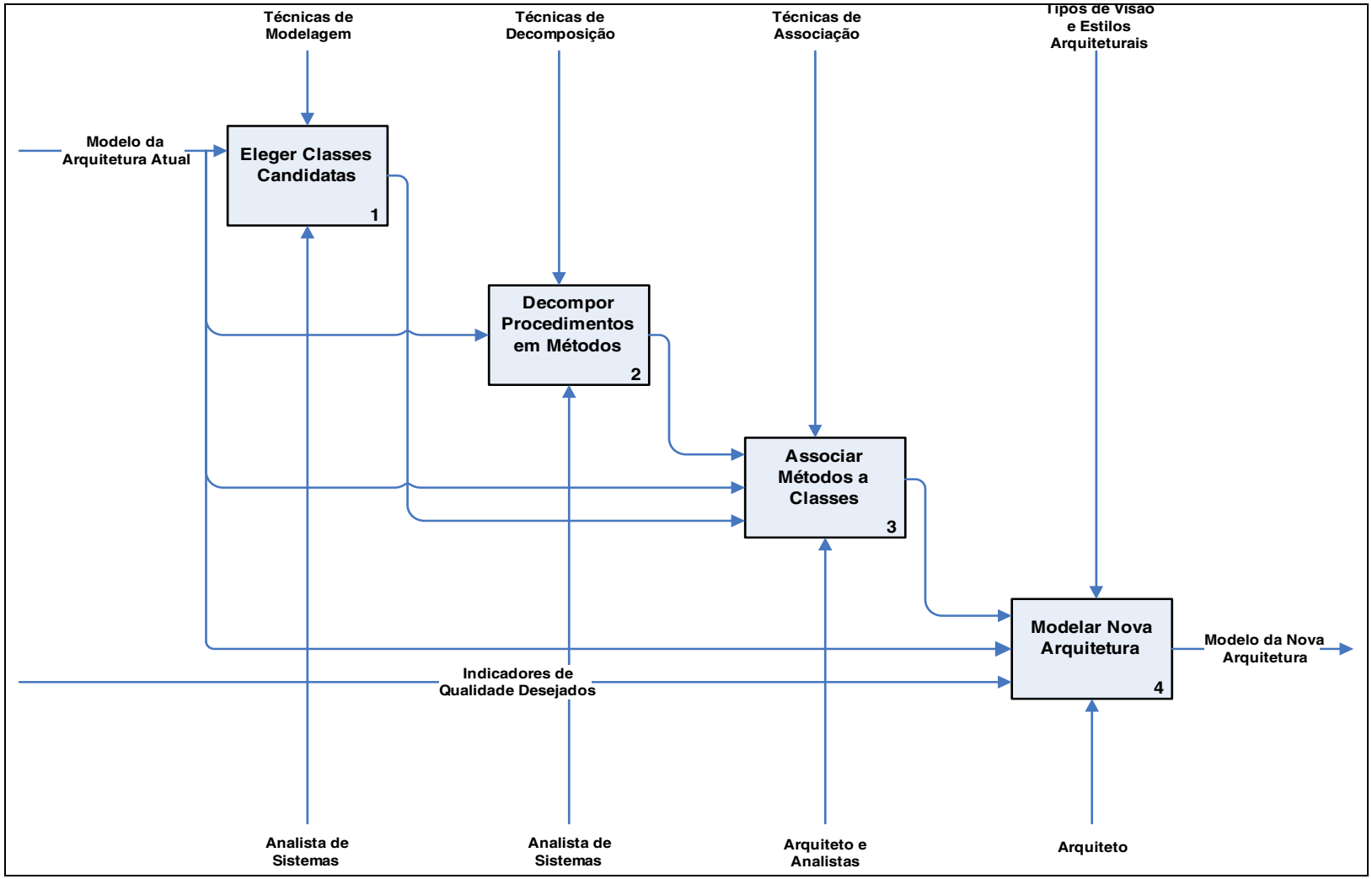

Figura 6.7 - Processo A3: Modelagem da Nova Arquitetura

Esta etapa é baseada nos processos migração de sistemas legados para OO pesquisados por De Lucia (1997), Bodhuin (2002) e Ying (2004). Ela visa atender os seguintes requisitos:

- Promover a mudança de paradigma e gerar um modelo que pode ser migrado incrementalmente a partir de uma arquitetura;
- Modelar as estruturas do sistema de forma a facilitar o atendimento dos indicadores desejados.

6.4.1 Eleger as Classes Candidatas

O objetivo dessa etapa é gerar um conjunto de classes candidatas a partir do modelo de dados gerados pela etapa Análise de dados.

Segundo De Lucia et al. (1997), vários métodos para identificação de um modelo orientado a objeto podem ser definidos dependendo do nível de granularidade desejado. No exemplo a seguir (quadro 6.9), são apresentadas algumas possibilidades de detalhamento a ser considerado pelo engenheiro de software.

Classe Candidata	**Estrutura de dados de origem**	**Tipo de estrutura de dados**
Classe A	Tabela A	Banco de Dados
Classe B	Estrutura de dados A	Variável global
Classe C	Tabela B	Banco de Dados
Classe D	Estrutura de dados B	Variável local
Classe E	Tabela D	Banco de Dados
Classe F	Estrutura de dados C	Variáveis de comunicação
Classe G	Tabela F	Banco de Dados
Classe H	Tabela G	Banco de Dados

Quadro 6.9 – Identificação de classes candidatas

6.4.2 Decompor Procedimentos em Métodos

Uma abordagem a ser considerada (De Lucia et al., 1997) é, enquanto a identificação de estados de objetos é centrada em depósitos de dados persistentes, os blocos de código do sistema legado são candidatos a implementar métodos destes objetos. Estes blocos podem ser programas batch inteiros, sub-rotinas ou grupos de sub-rotinas relacionados a uma chamada de procedimento (CALL).

Analogamente ao que ocorre na identificação de classes candidatas, componentes procedurais podem ser considerados em diferentes níveis de granularidade, ou seja, programas, sub-rotinas, porções de código, ou até mesmo um único comando da linha de código. O exemplo apresentado no quadro 6.10, apresenta algumas possibilidades que podem ocorrer:

- Os procedimentos P1 (programa PR1) , P2 (programa PR2) e o subprograma SP1 (programa PR3) serão substituídos pelo método candidato M1;
- Os trechos de procedimentos T1P3 (trecho 1 do procedimento 3 do programa PR3) e T1P1 (trecho 1 do procedimento 1 do programa PR4) serão substituídos pelo o método candidato M2;
- O procedimento 1 do será substituído por dois métodos candidatos M3 (trecho T1P1) e M4 (trecho T2P1);
- O método candidato M5 substituirá os trechos de procedimento T1P1 e T1P2 dos programas PR6 e PR7.

Método Candidato	Procedimento/Subprograma	Programa Origem
Método M1	Procedimento P1	Programa PR1
	Procedimento P2	Programa PR2
	Subprograma SP1	Programa PR3
Método M2	Trecho de procedimento T1P3	Programa PR3
	Trecho de procedimento T1P1	Programa PR4
Método M3	Trecho de procedimento T1P1	Programa PR5
Método M4	Trecho de procedimento T2P1	Programa PR5
Método M5	Trecho de Procedimento T1P1	Programa PR6
	Trecho de Procedimento T1P2	Programa PR7

Quadro 6.10 - Métodos Candidatos

Nessa etapa, teremos como artefatos gerados, um conjunto de porções de códigos (blocos) que serão candidatos a métodos no novo sistema. Os insumos a serem utilizados foram gerados na etapa Análise de procedimentos (tópico 6.3.3).

6.4.3 Associar Métodos a Classes

Uma vez que as classes e métodos candidatos foram definidos, o próximo passo consiste na associação dos métodos aos objetos (identificar qual objeto o método pertencerá). Para realizar essa associação dois aspectos importantes, resolução de conflitos e distribuição, devem ser considerados.

- Resolução de conflitos: a figura 6.8 (adaptada de Ying, 2004) ilustra os conflitos que podem ser encontrados nesta atividade. No exemplo abaixo: C1, C2 e C3 representam agrupamentos que são potenciais candidatos a se tornarem classes; A1 a A7 representam os atributos (tipos de dados); F1 a F4 representam as funções decompostas na etapa anterior e potenciais candidatos a métodos. No processo de segmentação do código pode ocorrer sobreposição de elementos entre

os grupos. Quando esta sobreposição ocorre na agregação dos atributos, pode ser que haja a necessidade de herança múltipla no relacionamento entre as classes candidatas. Quando ela ocorre nas funções, há a necessidade de uma análise para identificar qual a classe mais adequada para este futuro método.

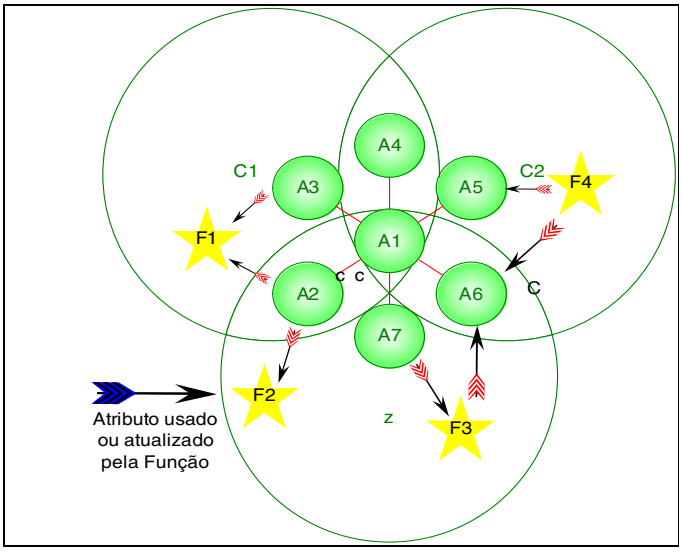

Figura 6.8 - Associação de métodos a objetos (Ying, 2004, p.4).

Para facilitar esta identificação é sugerido, por Ying (2004), um algoritmo baseado no modelo de aproximação de Markov. Este algoritmo avalia, dentre os possíveis objetos que um método pode pertencer, qual associação produzirá maior probabilidade de alcançar a qualidade desejada.

A qualidade desejada implica a parametrização de fatores como modularidade (envolvendo encapsulamento, coesão, acoplamento), complexidade (tamanho do componente, nível de herança de componentes, nível de complexidade do método, nível de complexidade da classe), entre outros.

- Distribuição: Programas como COBOL e RPG podem ser classificados como Batch ou On-line. Programas On-line possuem embutidos em si componentes que

gerenciam a interface com o usuário juntamente com componentes de domínio da aplicação. Programas batch podem possuir componentes de gerenciamento de interface com outros programas e componentes de domínio de aplicação. Componentes de interface com o usuário incluem instrução que comanda direta ou indiretamente o controle das instruções de entrada e saída. Desta forma, em um sistema cliente servidor, o cliente deve implementar principalmente a interface com o usuário e tem que controlar a interação e requisitar os serviços necessários para o servidor. Componentes de interface com o usuário são tipicamente migradas para aplicações que rodam no lado do cliente, enquanto que componentes que interagem com banco de dados são típicos candidatos a aplicações que rodam no lado do servidor. Componentes voltados para a lógica de negócio podem ficar tanto no lado do servidor, quanto do lado do cliente, dependendo da escolha do estilo. A figura 6.9 (adaptada de Cânfora et al., 1997) mostra as possibilidades mencionadas nesse parágrafo.

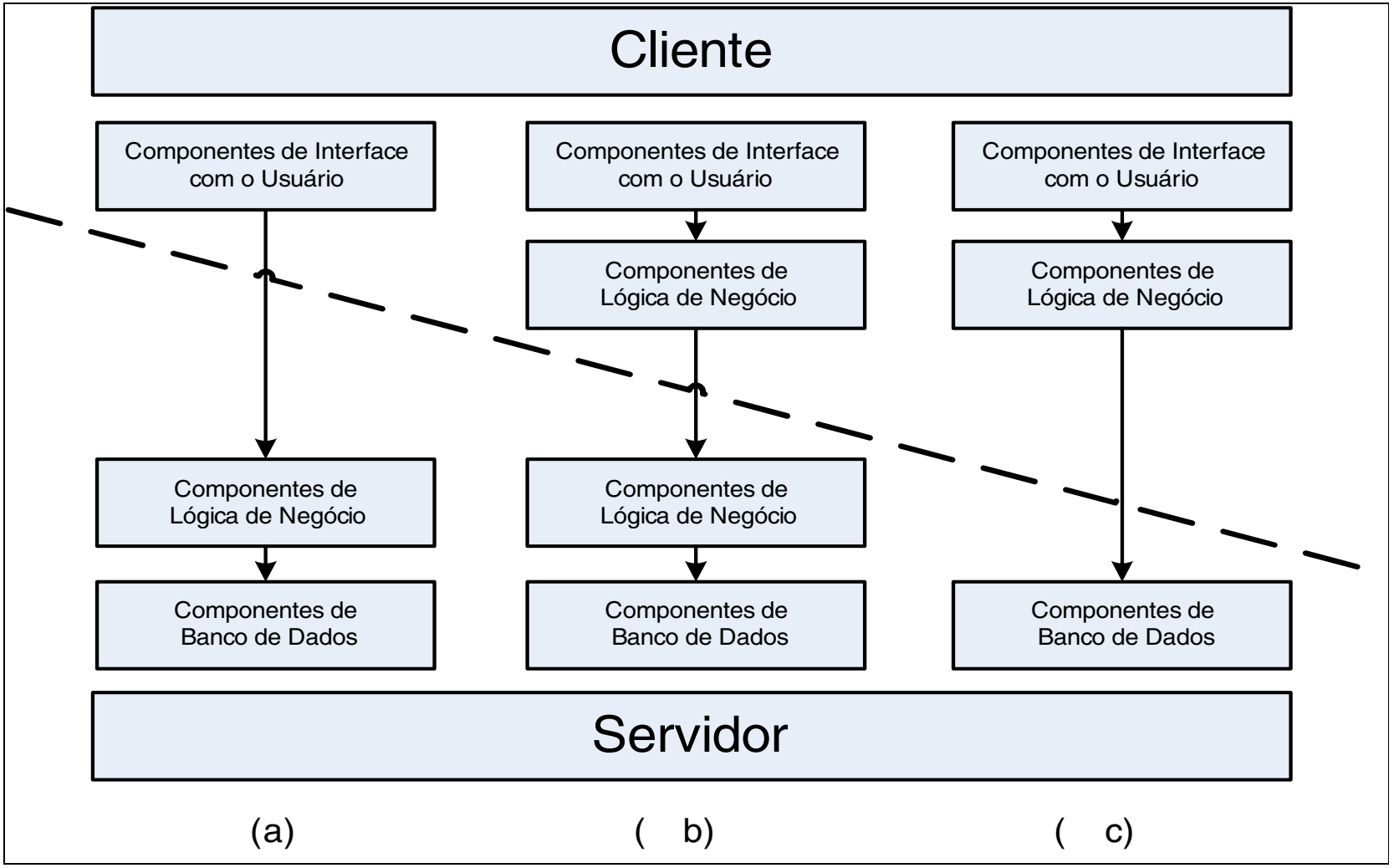

Figura 6.9 - Possibilidades de decomposição de código (Cânfora et al., 1997, p.2).

Os artefatos gerados nessa atividade constituirão de classes mapeadas com os respectivos objetos, conforme exemplo apresentado no quadro 6.11.

Classe	Método
Cartão	BloquearCartao
	CriarCartao
	ObterCartão
	RegistrarUsoCartao
	ValidarLimiteCartao
	ValidarSenhaCartão
Pedidos	AtualizarStatusPedido
	GerarPedidoFabricacao
	ObterCartõesFabricados
	ObterPedidoCartao
VinculoContaCartão	AssociarContaCartao
	ObterContasAssociadas
	ObterContasNãoAssociadas

Quadro 6.11 - Classes x Métodos

6.4.4 Modelar a Nova Arquitetura

A modelagem da nova arquitetura é uma fase onde os elementos levantados nas etapas anteriores serão organizados e complementados com mais visões para se obter o modelo arquitetural do sistema atual.

Uma vez solucionado o problema de mapeamento entre os elementos arquiteturais do sistema antigo procedural e do novo sistema OO, a idéia aqui é utilizar tipos de visão e estilos arquiteturais (proposto por Clements et al. 2003) para possibilitar que o software migrado seja criado numa estrutura mais robusta, flexível o suficiente para facilitar futuras manutenções.

Os indicadores de qualidade desejados e as priorizações serão importantes para a definição dos estilos arquiteturais a serem utilizados. Dessa forma, sugere-se a elaboração de uma planilha, semelhante ao exemplo apresentado no quadro 6.12, como forma de

identificar as visões e estilos arquiteturais necessários para ajudar a atingir as metas de qualidade definidas na etapa anterior (tópico 6.2).

Requisito	**Artefato arquitetural sugerido**
Flexibilizar customização de perfis de acesso a funcionalidades do sistema	Distribuição, Camada, Decomposição, Cliente Servidor
Tratamento de mensagens não previstas (tolerância a falhas)	Canos e Filtros, Processos Comunicantes, Cliente Servidor
Modificação de funcionalidade de forma a permitir correção de erros operacionais	Dados compartilhados, Cliente Servidor, Decomposição, Uso
Diminuir tempo de desenvolvimento a solicitações de mudança	Decomposição, Generalização, Implementação, Alocação de trabalho, Distribuição
Diminuir custo de desenvolvimento a solicitações de mudança	Distribuição, Implementação, Alocação de Trabalho
Melhorar tempo de resposta de determinadas transações on-line	Dados Compartilhados, Cliente Servidor, Peer-to-peer, Distribuição
Melhorar tempos do processamento Batch	Processos Comunicantes, Canos e Filtros, Dados Compartilhados
Diminuir quantidade de cancelamentos no Processamento Batch	Distribuição, Implementação, Processos Comunicantes, Canos e Filtros
Modificar a interface com o usuário de forma que operar com mais eficiência	Cliente Servidor, Decomposição, Implementação
Modificar a interface com o usuário de forma que assimilar o sistema mais rapidamente	Cliente Servidor, Decomposição, Implementação

Quadro 6.12 – Requisitos e Artefatos Sugeridos

Da mesma forma que na atividade Engenharia Reversa, sugere-se a construção de uma tabela que determine para cada envolvido o tipo de visão e o estilo arquitetural

necessário, conforme proposto por Clements et al. (2003) e cujo exemplo é apresentado no quadro 6.6.

O conjunto de artefatos gerados nessa etapa, além dos modelos OO obtidos a partir do código fonte, facilitará a tomada de ações e decisões arquiteturais para buscar as metas de qualidade definidas.

6.5 Implementação

Este processo (figura 6.10) consiste em determinar qual elemento arquitetural a ser migrado (componente que será transformado ou outra decisão arquitetural), implementá-lo e mensurá-lo, com o objetivo de avaliá-lo e gerar insumos para os próximos ciclos de migração.

Os insumos para este processo são: o modelo da nova arquitetura contemplando a mudança de paradigma de sistemas procedurais para OO, os indicadores de qualidade e as priorizações. Eles irão auxiliar na escolha dos elementos arquiteturais nas sucessivas migrações que serão realizadas até a transformação completa do sistema.

Após a implementação de cada elemento arquitetural será realizada uma avaliação da respectiva parte migrada, através da medição dos resultados obtidos em cada módulo, comparação com as medições do sistema atual e com indicadores de qualidade desejados. Os resultados obtidos na medição irão orientar a etapa de revalidar arquitetura que poderá ser modificada a fim de possibilitar a obtenção de indicadores melhores na próxima iteração.

O sub-processo de Implementação terá tantas iterações quantas forem necessárias até que todo o sistema tenha sido migrado.

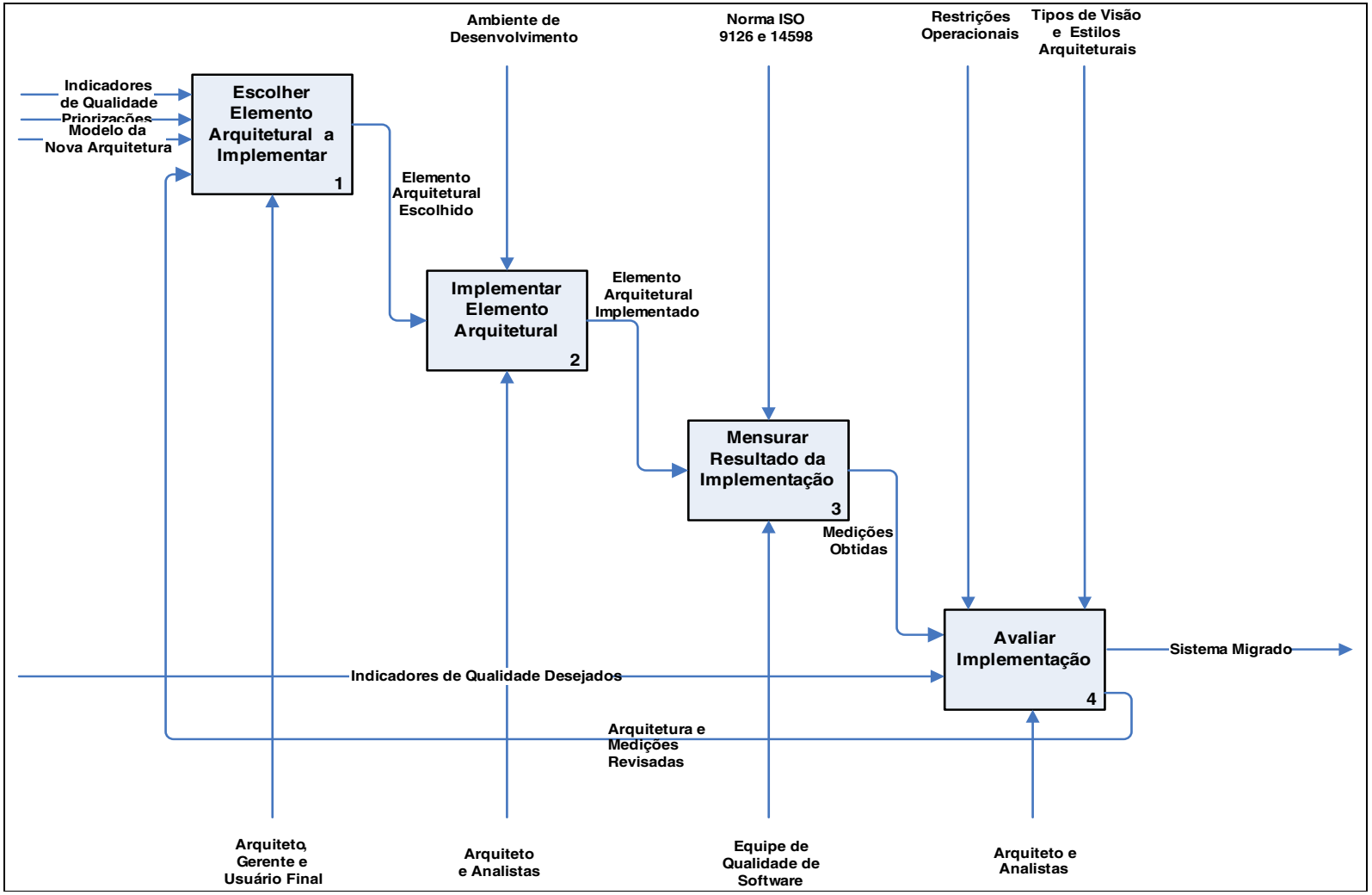

Figura 6.10 - Processo A4: Implementação

Os passos desta etapa foram propostos de forma a atender os seguintes requisitos:

- Permitir a inclusão de funcionalidades (atuais e novas) sem desagregar a arquitetura do sistema;
- Orientar a manutenção do novo sistema a partir dos requisitos de qualidade desejados.

6.5.1 Escolher Elemento Arquitetural a Implementar

Esta etapa consiste em decidir qual dos elementos arquiteturais do sistema deverá ser migrado. Esses elementos arquiteturais podem ser programas a serem convertidos e as decisões arquiteturais a serem realizadas.

No primeiro ciclo, essa escolha deverá ocorrer a partir do novo modelo de arquitetura elaborado na etapa anterior, dos indicadores de qualidade desejados e das priorizações. A partir do segundo ciclo, essa atividade passa a ser afetada pela implementação realizada nas iterações anteriores.

O resultado dessa atividade é a escolha de um ou mais elementos arquiteturais a ser implementado.

6.5.2 Implementar Elemento Arquitetural

Esta etapa consiste da realização de todos os procedimentos necessários para implementar objetos, ações e recursos que compõem os elementos arquiteturais. Esses procedimentos envolvem, no caso da migração, as etapas de desenvolvimento, implementação e teste unitários dos componentes.

6.5.3 Mensurar Resultado da Implementação

Esta etapa, além de mensurar os indicadores de qualidade de forma a permitir a sua posterior avaliação. Da mesma forma que na atividade "Mensurar Sistema Atual" (item 6.2.3), os cenários poderão ser utilizados como ferramenta de apoio (quadro 6.5) para registro dos resultados obtidos.

Ela deverá gerar um documento com o resultado das medições e será utilizada como entrada da próxima etapa.

6.5.4 Avaliar Implementação

Esta etapa consiste em avaliar se os indicadores de qualidade atendem os valores desejados, documentar os resultados, de forma a servir como orientação para futuras decisões arquiteturais, e revalidar as representações das visões e estilos arquiteturais que serão direcionados pelos indicadores de qualidade obtidos.

6.6. Conclusões do Capítulo

Atualmente existem muitos sistemas desenvolvidos sob o paradigma procedural. Estes programas foram desenvolvidos em linguagens como COBOL, RPG, entre outras, e por profissionais que foram formados (pela universidade ou pela empresa) a desenvolver sistemas sob o paradigma procedural. O número destes profissionais tem se reduzido com o passar do tempo e, na medida em que estas aplicações são desenvolvidas para atender novos requisitos, vão se tornando cada vez mais complexas, encarecendo o custo de desenvolvimento de novas solicitações.

Sistemas desenvolvidos sob a ótica da OO são mais fáceis de manter, segundo Battaglia et al. (1998), além de proverem facilidades de distribuição entre usuário e sistema (*browser*) e entre sistemas (*web services*).

Migrar um sistema legado para o paradigma OO é uma alternativa a ser considerada e a maioria das pesquisas realizadas nesta área foca muito mais na conversão do código do que na obtenção da especificação de uma arquitetura. Sistemas legados, com 20 ou 30 anos de uso são muito complexos e a sua completa especificação pode ser tão extensiva que compreender todos os seus aspectos através de um único enfoque é uma tarefa muito difícil.

Para tratar com esta complexidade, este trabalho propõe que esta migração seja realizada através de um processo de transformação de arquitetura com foco na mudança de paradigma de desenvolvimento. A utilização deste processo pode ser apoiada pela UML e provê uma especificação flexível e madura, na medida em que uma visão valida a outra e que o foco em aspectos diferentes permite o levantamento mais preciso de sistemas que geralmente apresentam grande complexidade.

CAPÍTULO 7
CONSIDERAÇÕES FINAIS

Objetivo do Capítulo

O objetivo deste capítulo é apresentar as considerações finais relativas ao trabalho, destacando a contribuição da qualidade e arquitetura de software no processo de migração para sistema orientado a objetos. Também são apresentadas contribuições e possíveis trabalhos futuros que podem ser gerados a partir dessa pesquisa.

7.1. Resumo das Contribuições do Trabalho

Os estudos realizados sobre manutenção e evolução de software (capítulo 2), reengenharia (capítulo 3), arquitetura de software (capitulo 4), qualidade de software (capítulo 5) e suas interligações constituem a base para uma migração de software que seja válida no sentido de melhoria de qualidade e, conseqüentemente, de produtividade.

O conjunto destas pesquisas permitiu a proposição de um processo de migração de arquitetura de sistema legado onde é possível ter evidências mensuráveis de melhoria da qualidade do produto resultante (novo sistema) e, além disso, permite o refinamento nas sucessivas iterações.

Pesquisas realizadas por De Lucia et al. (1997), Bodhuin et al. (2002) e Ying (2004) sugerem basicamente as seguintes etapas para migração de um código legado:

a) análise estática: o código fonte é analisado e toda informação útil para a próxima fase é extraída;

b) decomposição do programa: a idéia aqui é segmentar os programas em blocos de código;

c) abstração do modelo objeto: o modelo objeto do sistema é construído através das atividades de engenharia reversa (extração de um modelo de alto nível de abstração) realizadas nas etapas anteriores;

d) recodificação do próprio código legado: chamados por alguns autores de reengenharia, esta etapa consiste em recodificar o sistema legado na própria linguagem procedural;

e) encapsulamento: esta etapa consiste no encapsulamento de programas confeccionados através do código procedural (COBOL, RPG, entre outras) por métodos de objetos (denominados *wrappers*), confeccionados em uma linguagem OO. Mais uma vez, a estratégia é prover uma migração gradual, com riscos menores;

e) conversão para o código OO: esta etapa consiste na conversão total do código legado para o código orientado a objeto

As propostas que basicamente seguem as atividades relacionadas anteriormente focam muito mais na conversão do código do que na obtenção da especificação de uma arquitetura. Sistemas legados, com 20 ou 30 anos de uso são complexos e a sua completa especificação pode ser tão extensiva que compreender todos os seus aspectos é uma tarefa muito difícil. Para lidar com esta complexidade, a especificação de sistemas é geralmente decomposta através de processos de separação de conceitos para produzir um conjunto de especificações complementares, cada uma delas lidando com um aspecto específico de um sistema.

Dentre as propostas de migração de sistemas funcionais para orientado a objetos, este trabalho se diferencia por:

- Propor um processo de migração de paradigma focado em arquitetura ao invés de focar apenas em código;
- Usar os indicadores de qualidade como forma de evidenciar de forma mensurável melhorias no software que vai sendo desenvolvido aos poucos;
- Permitir que o sistema melhore a cada iteração, utilizando os resultados obtidos nas iterações anteriores como base para melhoria contínua das próximas.
- Processo de migração incremental que contempla elementos arquiteturais funcionais e não funcionais;

7.2. Trabalhos Futuros

O processo de migração do sistema legado funcional pode ser árduo, principalmente se considerarmos que a atividade de análise do código fonte tende a ser hercúlea nos casos de sistemas de grande porte, muito comum em grandes instituições financeiras.

Para reduzir estes problemas propõe-se, para apoio a este processo, o desenvolvimento de algoritmos de automação de transformação dos elementos de mais baixo nível, permitindo que o arquiteto foque em um nível mais abstrato, ou seja, mais próximo dos requisitos de alto nível que o software deve atender.

De forma análoga, a automação também pode incluir a medição dos indicadores de qualidade e sua utilização para as próximas iterações.

Tudo isso incluindo a utilização de modelos arquiteturais consagrados, o que contribui ainda mais para a robustez do sistema gerado, sem considerar que o software orientado a objetos permite maior potencial de melhoria de qualidade e produtividade.

7.3. Conclusões do Trabalho

A aplicação do processo de migração de sistemas legados funcionais obteve o resultado esperado, apesar de ter sido aplicado apenas em um projeto e utilizando um nível de granularidade baixo.

Algumas atividades mostraram certa dificuldade de execução, devido ao nível de detalhe que podem requerer (por exemplo, na decomposição de procedimentos em métodos).

Além disso, há de se levar em conta que a mudança de paradigma de desenvolvimento não é algo trivial se considerarmos que os profissionais envolvidos possuíam forte formação no paradigma procedural, mesmo com a sugestão que os profissionais com mais domínio do processo focassem em aspectos de mais alto nível.

Uma característica importante é que a aplicação do processo permitiu a recuperação de aspectos importantes do sistema, que antes estavam documentados apenas no código fonte, além de identificar diversos pontos de redução.

APÊNDICE A

APLICAÇÃO DO PROCESSO PROPOSTO

Objetivo do Apêndice

O objetivo deste capítulo é aplicar o processo de transformação de arquitetura de um sistema legado funcional em um sistema Orientado a Objetos.

A.1. Exemplo

Como exemplo foi escolhido um sistema onde o autor participou como analista e desenvolvedor.

O sistema escolhido para o exemplo trata do gerenciamento de cartões de débito e crédito cujas principais atribuições são:

- Administração dos cartões de crédito e débito;
- Monitoramento do crédito concedido pela instituição financeira decorrente das compras realizadas;
- Gerenciamento da compensação de valores entre a instituição, os lojistas e as empresas administradoras de cartão de crédito.

O contexto no qual este estudo se desenvolve é de uma empresa do mercado bancário de varejo em um País da América Latina. O sistema desta instituição tem a maior parte de seu processamento centralizado em computadores de médio porte (AS/400) e foi desenvolvido, em sua maior parte, através de linguagens orientadas a procedimentos como COBOL e CL, utilizando banco de dados DB2.

O software em operação tem mais de 10 anos de construção e atende satisfatoriamente os usuários. No entanto, tem sofrido várias manutenções decorrentes de alterações profundas na legislação, necessidades de melhoria de desempenho, criação de novas funcionalidades para atender à demanda do mercado, entre outras. Além disso, a empresa tem encontrado dificuldades em repor a mão de obra necessária para esta plataforma.

A experiência relatada decorreu da somatória de problemas acumulados que começaram a interferir no prazo para o desenvolvimento de novas funcionalidades de forma a responder rapidamente às necessidades da instituição financeira no sentido de manter sua competitividade no mercado bancário.

A.2 Avaliação do Sistema Atual

O objetivo desta atividade é identificar de forma sistemática os problemas apresentados pelo sistema e dar insumos para decidir sobre a viabilidade de migrar ou não o sistema para um novo paradigma.

A.2.1 Capturar Requisitos

Os requisitos de qualidade foram capturados através da utilização de cenários de qualidade. Foram gerados vários cenários de qualidade úteis para obter diversos indicadores do sistema atual.

A.2.2 Priorizar Requisitos

Os requisitos de qualidade foram priorizados e, colhidas as impressões de gerente de sistema, gestores e usuários, foram sugeridos os seguintes indicadores:

a) quantidade de cancelamentos no processamento batch por mês

b) tempo de resposta das transações on-line

c) quantidade de ocorrências no processamento on-line

d) percentual de anteprojeto definido x solicitações dos usuários

e) quantidade de horas solicitadas à fábrica de software

f) prazo orçado inicial x Prazo efetivamente cumprido

A.2.3 Mensurar Sistema Atual

Através destes indicadores procedeu-se com a mensuração dos mesmos, em relação ao sistema atual, que ocorreu através da compilação de relatórios de cancelamentos, envio de e-mails entre os envolvidos no sistema, definições de orçamentos entre outros. Estes valores foram compilados em uma planilha e discutidos entre os colaboradores a fim de obter um entendimento consensual.

A.2.4 Gerar Diagnóstico

A decisão sobre a migração foi baseada na medição destes indicadores e também em função da disponibilidade de analistas nas plataformas envolvidas. Notou-se que, no caso da instituição envolvida, havia uma clara tendência de aumento de funcionários com formação em desenvolvimento de aplicativos de baixa plataforma em detrimento da escassez de funcionários habituados a trabalhar na plataforma AS/400. Além disto, constatou-se também que a maior parte das inovações metodológicas, de software, entre outros, eram direcionados prioritariamente para o paradigma da orientação a objetos.

A.3 Engenharia Reversa

O objetivo deste processo é recuperar o estado atual da arquitetura e da implementação do aplicativo, através das visões do negócio e do projeto, proporcionando o entendimento sobre características do sistema. No trabalho proposto, sua realização se dará através das etapas de captura do processo de negócio, análise estática, análise do código fonte e extração do modelo arquitetural.

A.3.1 Captura do Processo de Negócio

Com o objetivo de obter um modelo de processo de negócio, foram realizadas reuniões com os usuários, analistas de negócio e desenvolvedores da aplicação. Através destas reuniões foram extraídos os processos de negócio descritos na quadro A.1.

Processo	**Descrição**
Administrar cartão	Neste processo são realizados novos pedidos, renovações, substituições, bloqueio e desbloqueio de utilização, validação, troca de categoria e cancelamento de cartões de débito e crédito.
Monitorar uso do cartão	Neste processo estão envolvidos a emissão e envio da fatura para os clientes, acompanhamento do pagamento, emissão de informações ao banco central, verificação de devedores duvidosos, emissão de informações para *credit scoring*.
Gerenciar Compensação das operações	Neste processo estão envolvidos o pagamento de comissões às administradoras relativos a compras realizadas pelos clientes, o recebimento de pagamentos realizados pelos clientes, o repasse de valores aos lojistas, captura da comissão do banco.

Quadro A.1 - Processos apoiados pelo sistema

Como complemento à descrição dos processos, foi gerado um diagrama em IDEF0, posteriormente validado com os usuários. Este diagrama é mostrado na figura A.1.

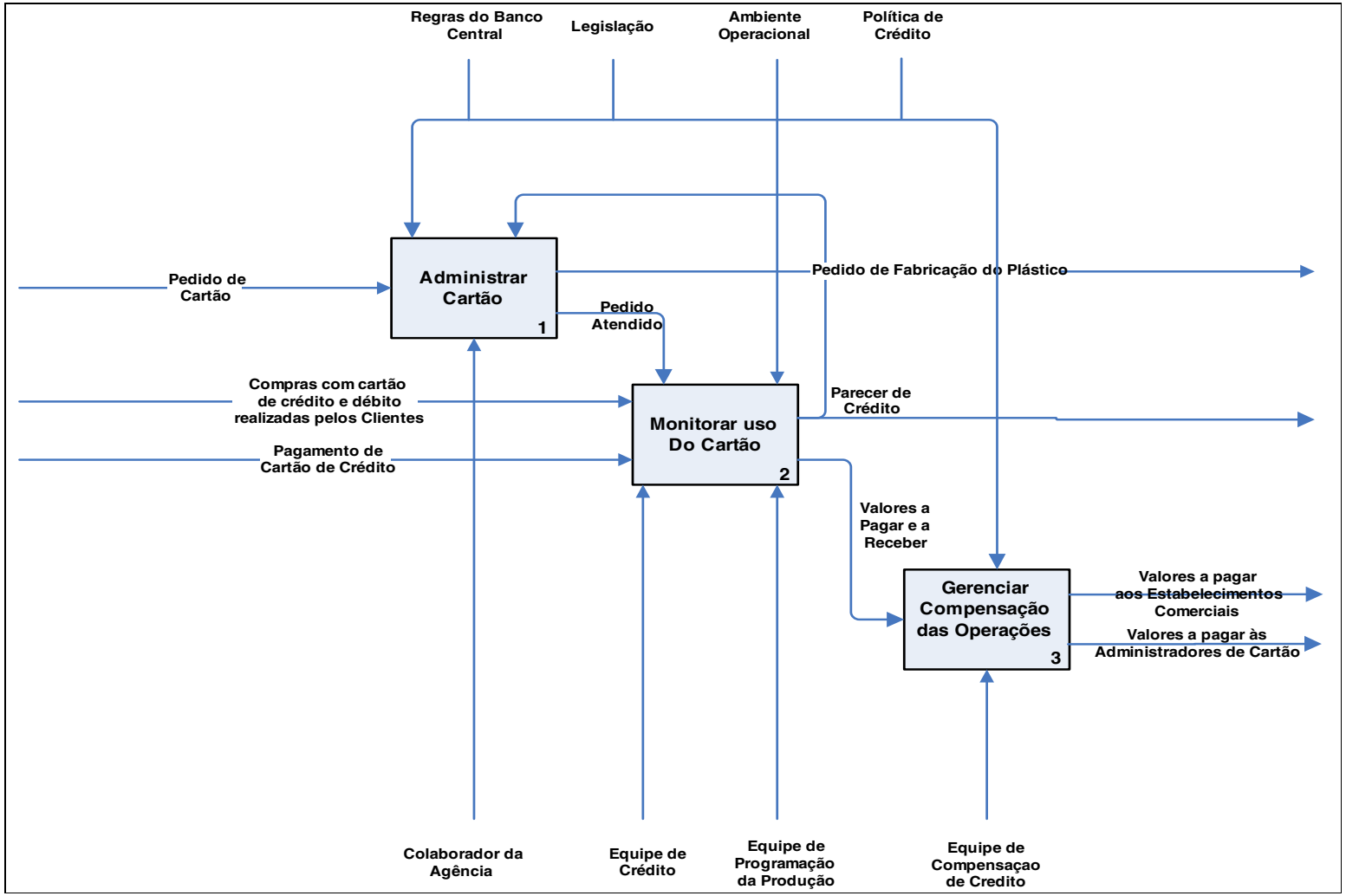

Figura A.1 - Processo A0: Sistema de Cartão de Crédito e Débito

Por se tratar de um sistema relativamente grande e os processos apresentarem fraco acoplamento entre si, focou-se apenas o processo “Administrar Cartão”. Através de novas reuniões com os usuários obteve-se o detalhamento deste processo, conforme apresentado no quadro A.2.

Processo	Descrição
Pedir Cartão	Processo responsável pela solicitação de fabricação do cartão de crédito, acompanhamento do envio do mesmo para o cliente e validação das regras para sua solicitação junto às normas da instituição, leis vigentes no país e via verificação de cadastro de clientes devedores.
Substituir Cartão	Este processo é responsável pela substituição do cartão seja ele motivado pelo vencimento do plástico, troca de categoria ou troca da administradora de cartão. Nos dois últimos casos o pedido pode ter sido originado pela cliente ou pela instituição (desde que aprovado pelo cliente), em função de sua pontuação no sistema.
Bloquear Cartão	Neste processo é realizado o bloqueio do cartão, seja a pedido do cliente, a pedido da instituição ou a partir de solicitações originadas de processos de monitoração de clientes devedores.
Validar Cartão	Processo responsável pela validação dos cartões de débito e crédito e sua utilização em pontos de venda, caixas eletrônicos, internet e outros canais de venda da instituição.

Quadro A.2 - Processo Administrar Cartão

Com objetivo de prover um refinamento da descrição deste processo, também foi gerado um diagrama em IDEF0, mostrado na figura A.2.

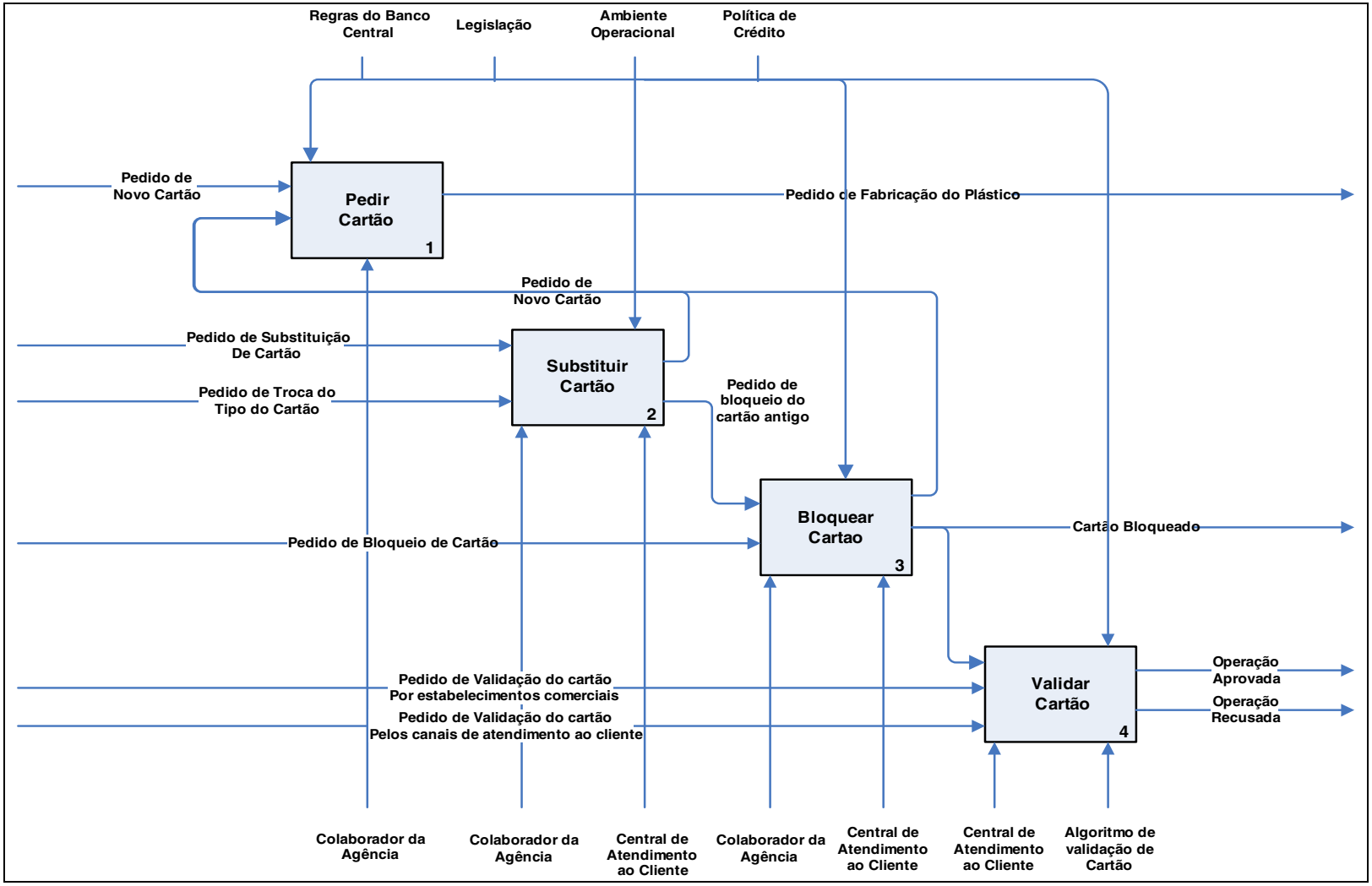

Figura A.2 - Processo A1: Administrar Cartão

A.3.2 Análise dos Dados

Dando continuidade à aplicação deste processo, procedeu-se ao levantamento das principais estruturas de dados utilizadas pelo sistema. Estas informações foram extraídas do banco de dados (DB2/400), arquivos, variáveis locais e globais de programas e subprogramas.

Dado a grande quantidade de estrutura de dados e respectivos atributos, a título de visualização, as informações foram apresentadas no nível de domínio, exemplificando apenas aquelas consideradas como principais. Na figura A.3, foi utilizado o diagrama de entidade e relacionamento (DER) devido à familiaridade entre os analistas envolvidos no processo.

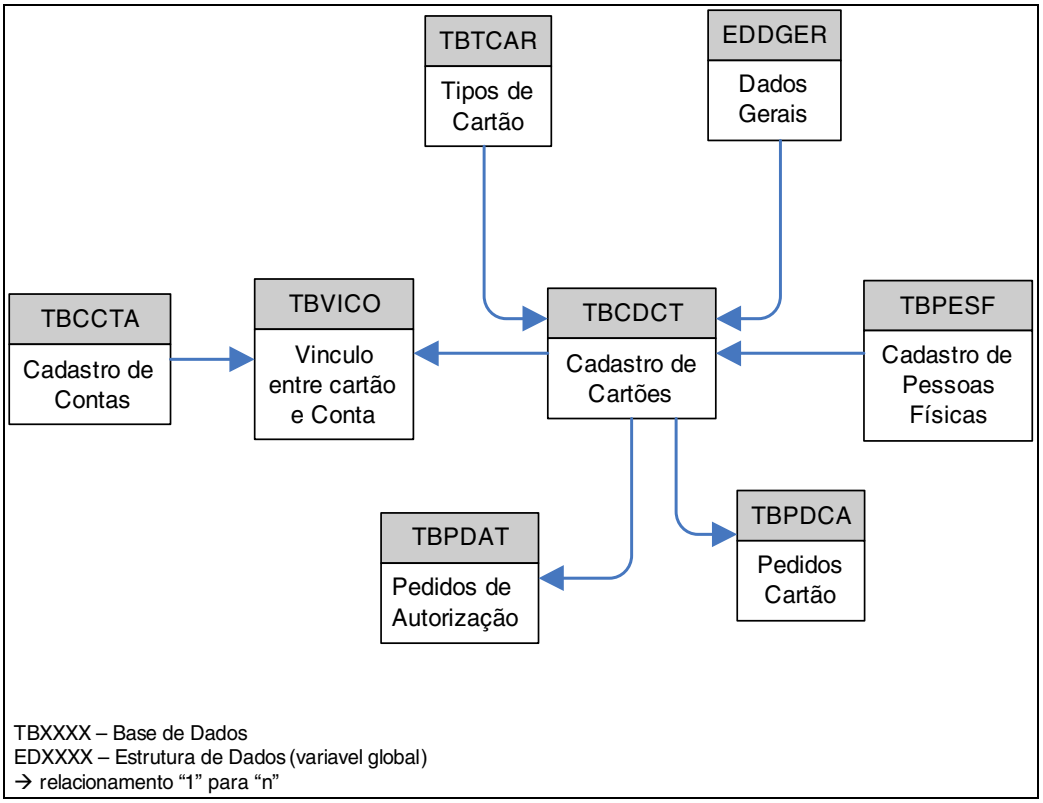

Figura A.3 - Diagrama Entidade x Relacionamento: Subsistema Administrar Cartão

A.3.3 Análise do Código Fonte

Nesta etapa prosseguiu-se com a análise do código fonte, onde foram identificados procedimentos e chamadas a subprogramas ou funções. No quadro A.3 encontra-se relacionados alguns destes procedimentos.

Programa	**Tipo**	**Procedimento/Subprograma**
Pedir Cartão	On-line	Obter dados Gerais
		Obter tipos de cartão
		Obter dados do Cliente
		Obter contas do cliente
		Associar Conta ao Cartão
		Inserir pedido de cartão
Solicitar fabricação do cartão	Bach	Obter dados gerais
		Obter pedidos de cartão
		Obter dados de cada cliente
		Obter dados do tipo do cartão
		Inserir dados no arquivo de fabricação
		Atualizar Pedido de Cartão
Recepcionar retorno da fabricação	Bach	Obter dados Gerais
		Obter cartões fabricados
		Obter pedido de cartão
		Inserir dados no cadastro de cartões
		Atualizar pedido de cartão
Substituir Cartão	On-line	Obter dados Gerais
		Obter dados do Cadastro de Cartões
		Obter tipos de cartão
		Obter contas do cliente
		Associar Conta ao novo Cartão
		Inserir pedido do novo cartão
		Bloquear cartão substituído

Quadro A.3 - Programas do Subsistema Administrar Cartão

Programa	Tipo	Procedimento/Subprograma
Validar Senha de Cartão	On-line	Obter dados Gerais
		Obter dados do Cadastro de Cartões
		Validar Senha do Cartão
		Validar Limite de Operação
		Gerar *log* de uso do cartão
Associar Conta ao Cartão	On-line	Obter Dados Gerais
		Obter Contas Ativas
		Obter Contas não associadas ao cartão
		Obter associações de contas assinaladas pelo usuário
		Associar Contas ao Cartão
Bloquear Cartão	On-line	Obter dados Gerais
		Obter dados do Cadastro de Cartões
		Bloquear cartão atual
		Pedir novo cartão

Quadro A.3 - Programas do Subsistema Administrar Cartão (continuação)

A.3.4 Extração do Modelo Arquitetural

Nesta etapa do processo foram organizados os elementos levantados nas etapas anteriores para obter o modelo arquitetural do sistema atual. Com o objetivo de melhor compreender o sistema e permitir a sugestão de uma nova arquitetura, foram utilizados os tipos de visão e estilos arquiteturais propostos por Clements et al. (2003).

A definição dos estilos arquiteturais a serem considerados foi elaborada com a definição dos seguintes envolvidos:

- Gerente de Projeto;
- Analistas de Sistemas;
- Equipe de Qualidade;
- Analista de Produção;

- Gestores do sistema.

Para atender as necessidades de cada envolvido, foram sugeridos os estilos arquiteturais apresentados no quadro A.4.

	Tipos de Visão				
	Módulos		**Componentes e Conectores**	**Alocação**	
Envolvido	Decomposição	Uso	Canos e Filtros	Distribuição	Atribuição de Trabalho
Gerente de projeto	D	-	G	G	D
Analista de sistemas	D	-	D	-	-
Equipe de Qualidade	D	G	D	-	G
Analista de Produção	G	G	G	D	-
Gestores do Sistema	G	-	G	G	-
Legenda: D - informação detalhada; M - grau médio de detalhamento; G - informação geral					

Quadro A.4 - Estilos arquiteturais sugeridos

Considerou-se para cada envolvido o nível de detalhamento necessário. Neste trabalho, no entanto, foi considerado somente o nível geral de detalhamento.

A.3.4.1 Visão Módulos

Nesta etapa foram considerados os estilos arquiteturais decomposição e uso. O objetivo do utilização desse tipo de visão é mostrar o sistema como um conjunto de unidades de implementação, cada um com um conjunto de responsabilidades.

No estilo arquitetural de decomposição de módulos o código fonte foi decomposto de forma *top-down*. A figura A.4 apresenta o módulo de administração de cartão e seus sub-módulos.

No estilo arquitetural de “uso” procurou-se identificar a dependência entre os módulos e a possibilidade de desenvolvimentos paralelos. A figura A.5 apresenta o uso deste estilo no nosso exemplo.

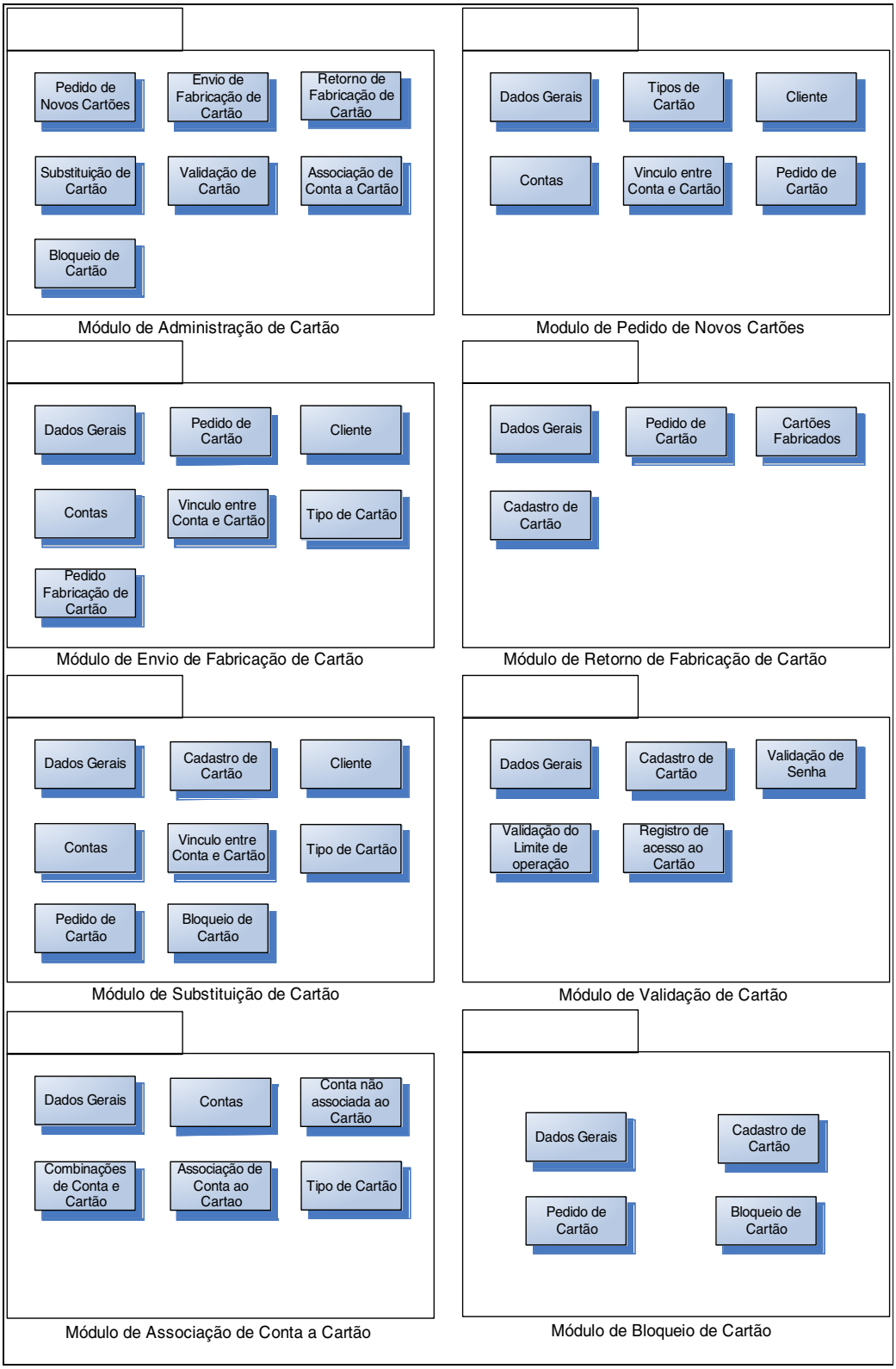

Figura A.4 - Decomposição do Subsistema Administrar Cartões

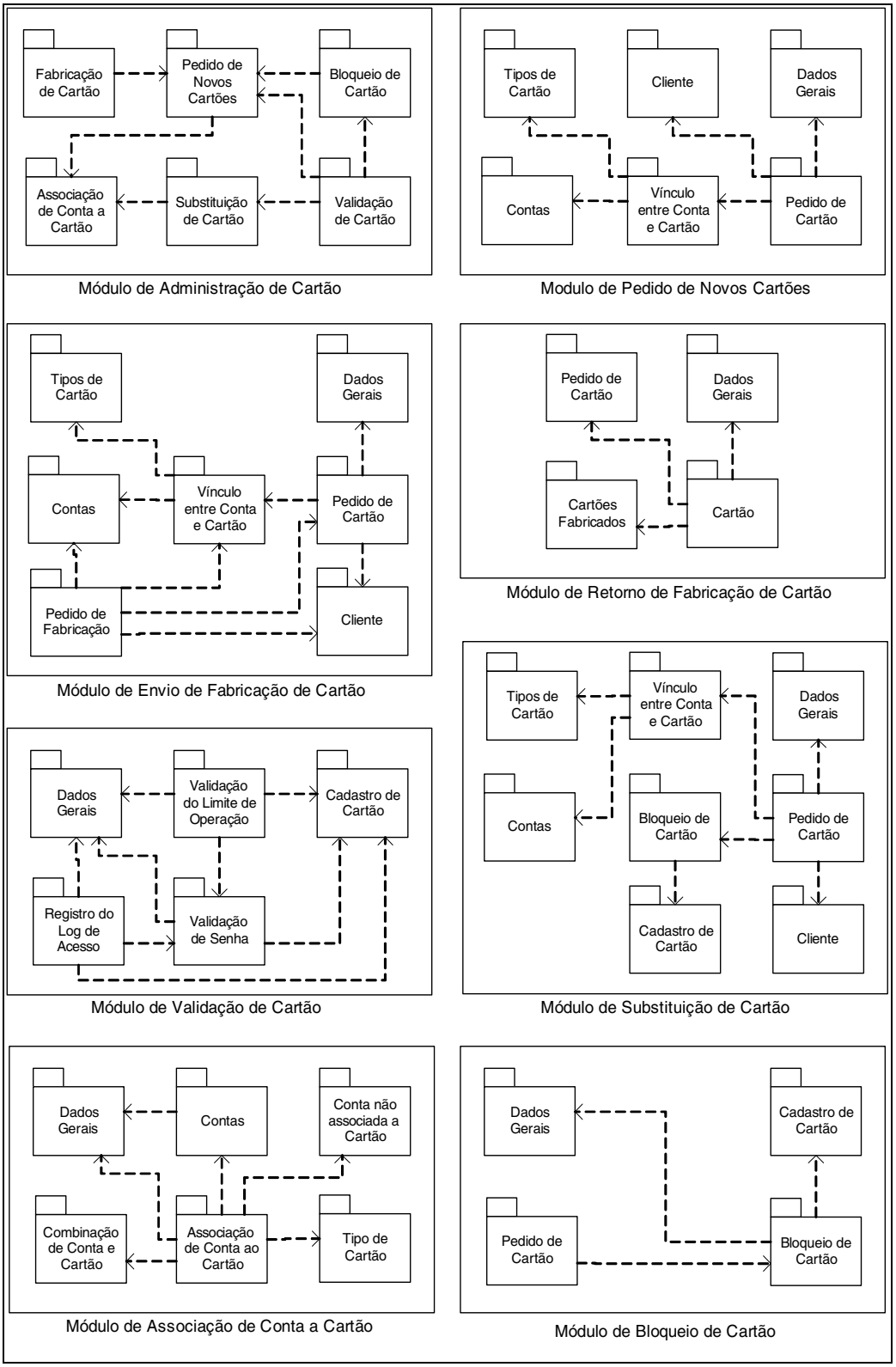

Figura A.5 -Relação entre os módulos do Subsistema Administrar Cartões

A.3.4.2 Visão Componentes e Conectores

Nesta etapa foram considerados o estilo arquitetural canos e filtros. Este tipo de visão trabalha com elementos existentes em tempo de execução e com elementos de interação. O objetivo de se utilizar este tipo de visão é mostrar o sistema do ponto de vista do processamento batch e on-line.

No estilo arquitetural “canos e filtros”, ocorre a transmissão de dados de um componente para outro. Estes componentes são projetados para esperar dados de entrada num determinado formato e produzir dados de saída para o próximo filtro. Na figura A.6 é visualizada a aplicação deste estilo no exemplo considerado.

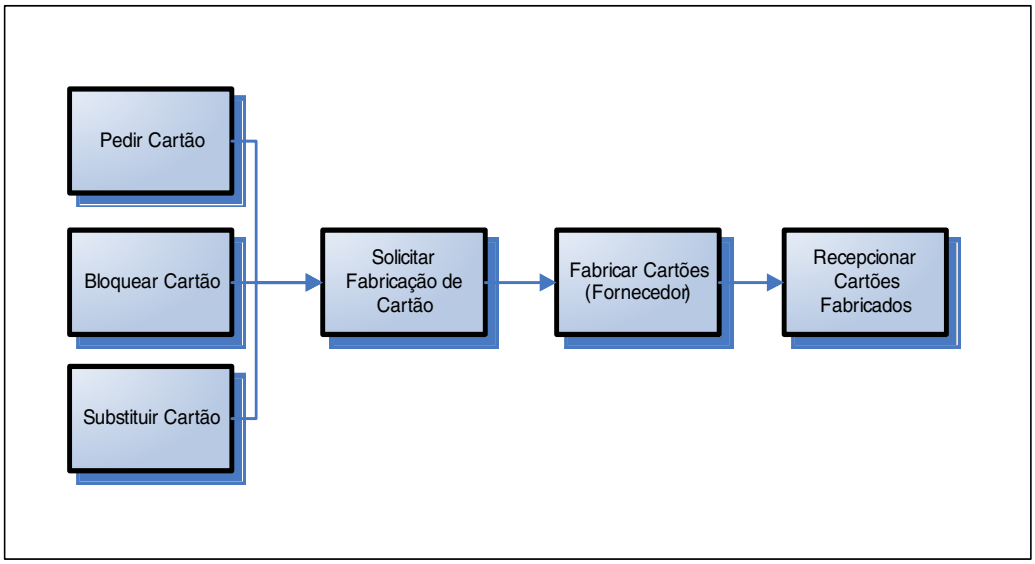

Figura A.6 - Canos e Filtros: Subsistema Administrar Cartões

A.3.4.3 Visão Alocação

A visão alocação mostra o software e seus relacionamentos com o hardware, sistemas de arquivos e equipes de desenvolvimento. Para demonstrar a alocação dos módulos de software às equipes de desenvolvimento, foi utilizado o estilo arquitetural atribuição de trabalho, conforme apresentado no quadro A.5.

Sistema	**Programa**	**Responsável**
Administrar Cartões	Pedir Cartão	Analista A
	Solicitar fabricação do cartão	Analista B
	Recepcionar retorno da fabricação	Analista B
	Substituir Cartão	Analista A
	Validar Senha de Cartão	Analista B
	Associar Conta ao Cartão	Analista A
	Bloquear Cartão	Analista A

Quadro A.5 - Atribuição de trabalho: Subsistema Administrar Cartões

A.4. Modelagem da Nova Arquitetura

Através dos elementos arquiteturais levantados nas etapas anteriores, será modelada a nova arquitetura do sistema. Nesta etapa foi considerado todo o sistema envolvido sem nos preocupar qual parte será transformada ou não. Nas etapas a seguir está descrito o mapeamento entre os elementos arquiteturais do sistema legado e o do sistema a ser construído.

A.4.1 Eleger as Classes Candidatas

Para a identificação de um modelo orientado a objetos, vários métodos podem ser definidos. Nesse exemplo, a fim de evitar a identificação de uma excessiva quantidade de classes candidatas, foram considerados somente os dados persistentes. O quadro A.6

apresenta a relação de classes candidatas obtidas a partir do modelo entidade-relacionamento apresentado na figura A.3.

Classe Candidata	Estrutura de dados de origem	Tipo de estrutura de dados
TipoCartão	Tipo de Cartão	Tabela SQL/400
DadosGerais	Dados Gerais	Variavel global
Conta	Cadastro de Contas	Tabela SQL/400
VinculoContaCartão	Vinculo entre Contas e Cartão	Tabela SQL/400
Cartão	Cadastro de Cartões	Tabela SQL/400
Pessoa	Cadastro de Pessoa Física	Tabela SQL/400
PedidoAutorização	Pedidos de Autorização	Tabela SQL/400
PedidoCartão	Pedidos de Cartão	Tabela SQL/400

Quadro A.6 - Classes Candidatas: Subsistema Administrar Cartões

A.4.2 Decompor Procedimentos em Métodos

Enquanto a identificação das classes candidatas foi baseada nos depósitos de dados, a identificação dos métodos candidatos foi obtida a partir de procedimentos, sub-rotinas ou grupos de sub-rotinas relacionados a uma chamada de procedimento. Analogamente ao que ocorreu na identificação de classes candidatas, optou-se por identificar os componentes procedurais em um nível menor de detalhamento, conforme apresentado no quadro A.7.

Método Candidato	Procedimento/Subprograma	Programa Origem
AssociarContaCartao	Associar Conta ao Cartão	Pedir Cartão
	Associar Conta ao novo Cartão	Substituir Cartão
	Associar Contas ao Cartão	Associar Conta ao Cartão
AtualizarStatusPedido	Atualizar Pedido de Cartao	Solicitar fabricação do cartão
	Atualizar pedido de cartão	Recepcionar retorno da fabricação
BloquearCartao	Bloquear cartão atual	Bloquear Cartão
	Bloquear cartão substituído	Substituir Cartão

Quadro A.7 - Métodos Candidatos: Subsistema Administrar Cartões

Método Candidato	**Procedimento/Subprograma**	**Programa Origem**
CriarCartao	Inserir dados no cadastro de cartões	Recepcionar retorno da fabricação
	Inserir pedido de cartão	Pedir Cartão
	Inserir pedido do novo cartão	Substituir Cartão
	Pedir novo cartão	Bloquear Cartão
GerarPedidoFabricacao	Inserir dados no arquivo de fabricação	Solicitar fabricação do cartão
ObterPedidoCartao	Obter pedido de cartão	Recepcionar retorno da fabricação
	Obter pedidos de cartão	Solicitar fabricação do cartão
ObterCartão	Obter dados do Cadastro de Cartões	Substituir Cartão
	Obter dados do Cadastro de Cartões	Validar Senha de Cartão
	Obter dados do Cadastro de Cartões	Bloquear Cartão
ObterCartoesFabricados	Obter cartões fabricados	Recepcionar retorno da fabricação
ObterCliente	Obter dados de cada cliente	Solicitar fabricação do cartão
	Obter dados do Cliente	Pedir Cartão
ObterConta	Obter Contas Ativas	Associar Conta ao Cartão
	Obter contas do cliente	Pedir Cartão
	Obter contas do cliente	Substituir Cartão
ObterContasAssociadas	Obter associações de contas assinaladas pelo usuário	Associar Conta ao Cartão
ObterContasNaoAssociadas	Obter Contas não associadas ao cartão	Associar Conta ao Cartão

Quadro A.7: Métodos Candidatos - Subsistema Administrar Cartões (continuação)

Método Candidato	**Procedimento/Subprograma**	**Programa Origem**
ObterDadosGerais	Obter dados Gerais	Todos
	Obter dados gerais	Solicitar fabricação do cartão
	Obter dados Gerais	Recepcionar retorno da fabricação
	Obter dados Gerais	Substituir Cartão
	Obter dados Gerais	Validar Senha de Cartão
	Obter Dados Gerais	Associar Conta ao Cartão
	Obter dados Gerais	Bloquear Cartão
ObterTipoCartao	Obter dados do tipo do cartão	Solicitar fabricação do cartão
	Obter tipos de cartão	Pedir Cartão
	Obter tipos de cartão	Substituir Cartão
RegistrarUsoCartao	Gerar *log* de uso do cartão	Validar Senha de Cartão
ValidarLimiteCartao	Validar Limite de Operação	Validar Senha de Cartão
ValidarSenhaCartao	Validar Senha do Cartão	Validar Senha de Cartão

Quadro A.7: Métodos Candidatos - Subsistema Administrar Cartões (continuação)

A.4.3 Associar Métodos a Classes

Esta etapa do processo consiste na associação dos métodos aos objetos (identificar qual objeto o método pertencerá). O quadro A.8 apresenta esta associação para o exemplo proposto.

Classe	Método
Cartão	BloquearCartao
	CriarCartao
	ObterCartão
	RegistrarUsoCartao
	ValidarLimiteCartao
	ValidarSenhaCartao
Conta	ObterConta
DadosGerais	ObterDadosGerais
PedidoCartao	AtualizarStatusPedido
	GerarPedidoFabricacao
	ObterCartõesFabricados
	ObterPedidoCartao
Cliente	ObterCliente
TipoCartao	ObterTipoCartao
VinculoContaCartao	AssociarContaCartao
	ObterContasAssociadas
	ObterContasNaoAssociadas

Quadro A.8 - Classes x Métodos: Subsistema Administrar Cartões

A.4.4 Modelar a Nova Arquitetura

Nesta etapa foi realizada a transformação da arquitetura, uma vez solucionado o problema de mapeamento entre os elementos arquiteturais do sistema (módulos) antigo procedural e do novo sistema OO. Focou-se na visão de módulo por se tratar da parte mais afetada neste processo de mudança de paradigma. A figura A.8 mostra como ficou o novo sistema.

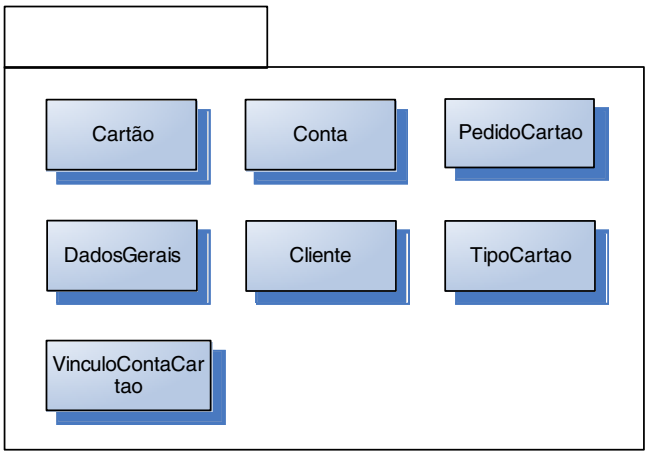

Figura A.8 - Decomposição do Novo Subsistema Administrar Cartão

Também foi utilizada a visão de uso (figura A.9) para refletir os novos relacionamentos entre os módulos.

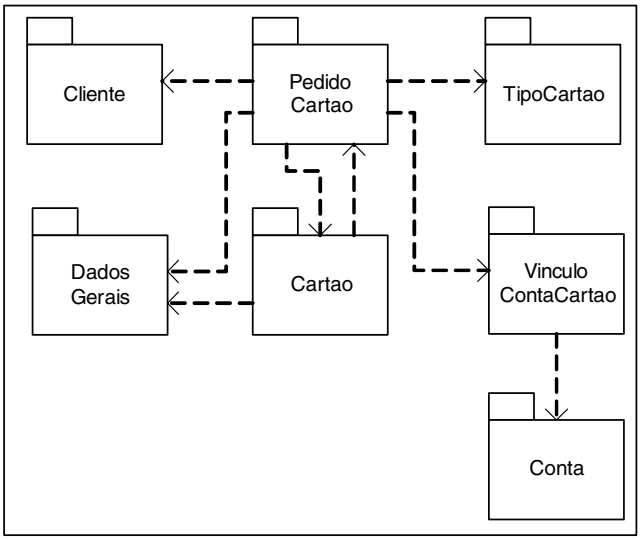

Figura A.9 - Relação de Uso: Novo Subsistema Administrar Cartões

A.5 Implementação

Com base no modelo de arquitetura OO gerado na etapa anterior e os indicadores de qualidade, os envolvidos no processo optaram por escolher todo o módulo de administração de cartão (pois não era considerado muito grande). A partir da implementação deste módulo pode-se avaliar os indicadores produzidos por esta nova situação a fim de verificar se houve ganhos de qualidade.

Após a implementação de cada módulo transformado foi realizada uma avaliação da respectiva parte migrada, através da utilização de indicadores de qualidade definidas no processo “avaliar sistema atual”, a fim de verificar se houve melhoria de desempenho e orientar decisões arquiteturais que possam ser tomadas nas próximas iterações.

A. 6. Conclusões do Apêndice

Neste apêndice foi aplicado o processo em sistema desenvolvido pelo autor e foram identificados vários pontos de melhoria dentre os quais se destacam a redução de redundâncias, maior robustez e simplicidade na arquitetura.

Os resultados demonstram que a utilização das técnicas e práticas propostas pelo processo permite a redução da complexidade do sistema, melhoria da qualidade, além de permitir uma melhor integração do sistema com novas tecnologias.

LISTA DE REFERÊNCIAS

BACHMANN, F.; BASSS, L.; KLEIN, M.; SHELTON, C. Designing Software Architectures to Achieve Quality Attribute Requirements. **IEEE Software**, v. 152, n.4, p. 153-165, August 2005.

BASS, Lens; CLEMENTS, Paul; KAZMAN, Rick. **Software Architecture in Practice**. Addison-Wesley, 2003.

BATTAGLIA, M.; GIANCARLO, S.; FAVARO, J. RENAISSANCE: A Method to Migrate from Legacy to Immortal Software Systems. In: Second Euromicro Conference on Software Maintenance and Reengineering, 1998, Florence. **Proceedings.** 1998, p.197-200.

BODHUIN, Thierry; GUARDABASCIO, Enrico; TORTORELLA, Maria. Migrating COBOL Systems to the WEB by using the MVC Design Pattern. **IEEE Software**, p.329-338, November 2002.

BOSCH, J.; MOLIN, P. Software architecture design: evaluation and transformation. **IEEE Software**, p.4-10, March 1999.

CÂNFORA, G.; CIMITILE, A.; DE LUCIA, A.; DE LUCCA, G.A.. Decomposing Legacy Programs: a First Step Towards Migrating to Client-Server Platforms. **IEEE Software**, p.136-144, June 1998.

CÂNFORA, G.; FASOLINO, A. R.; FRATTOLILLO, G.; TRAMONTANA, P. Migrating Interactive Legacy Systems To Web Services. In: 10th European Conference on Software Maintenance and Reengineering, 2006. **Proceedings.** 2006, pp10.

CLEMENTS, P.; GARLAN, D.; LITTLE, R.; NORD, R.; STAFFORD, J. Documenting Software Architectures: Views and Beyond. **Proceedings of the 25th International Conference on Software Engineering** (ICSE.03) 0270-5257/03, 2003 IEEE.

COCKBURN, A. **Writing Effective Use Cases.** Addison Wesley. Boston, 2000.

DE LUCIA, A.; DI LUCCA, G. A.; FASOLINO, A. R.; GUERRA, P.; PETRUZZELLI, S. Migrating Legacy Systems towards Object-Oriented Platforms. In: International Conference on Software Maintenance and Reengineering, 1998, Salerno**,** Italy. **Proceedings.** 1997, p.122-129.

DUCASSE, S. **Reengineering Objetct-Oriented Applications.** 2003. 157 f. Habilitação para condução de pesquisas – Université Pierre et Marie Curie, Paris, França, 2003.

DUCASSE, S.; DEMEYER, S.; NIERSTRASZ, O. **A pattern language for reverse engineering.** In: European Conference on Pattern Languages of Programming and Computing, 5., 1999, Alemanha. Proceedings of the 5th European Conference on Pattern Languages of Programming and Computing. Konstanz Universitätsverlag, 1999. p. 189-208.

GJÖRWELL, Daniel; HANGLUND, Staffan; SANDEL, Daniel. **Reengineering and Reengineering Patterns.** Department of Computer Science and Engineering Mälardalens Högskolas, 2002.

MAGALHÃES, DIMAS Ribeiro; ARAKAKI, Reginaldo. **Um Processo de Melhoria Contínua da Arquitetura de Software Direcionado por Indicadores de Qualidade.** 2005 IPT.

NBR ISO/IEC 14598-1, **Tecnologia da Informação – Avaliação de Produto de Software. Parte 1: Visão Geral,** 2001.

NBR ISO/IEC 9126-1, **Engenharia de Software – Qualidade de produto. Parte 1: Modelo de Qualidade,** 2003

PANAS, T.; LÖWE, W.; ABMAN, U. Towards the Unified Recovery Architecture for Reverse Engineering. **Proceds of the International Conference on Software Engineering Research and Practice.** Nevada, EUA, vol. 2, p. 854-860, jun. 2003.

PFLEEGER, S. L., **Engenharia de Software: Teoria e Prática,** Tradução: Dino Franklin Ed. Prentice Hall. São Paulo, 2004.

PRESSMAN, R. S. **Software Engineering: A Practitioner's Aproach,**. McGraw-Hill, 2002.

SOUZA, Rodrigo Álvares; ARAKAKI, Reginaldo. **Um Processo de Transformação de Arquitetura de Sistemas Legados Baseados em Reengenharia.** 2004 POLI-USP.

YING, Z.; KONTOGIANIS, K. Migration to Object Oriented Platforms: A State Transformation Approach**. IEEE Software**, p.530-539, 2002.

YING, Z. Incorporating Quality Requirements in Software Migration Process. **IEEE Software,** p.175-185, sept. 2003.

YING, Z. Incremental Quality Driven Software Migration to Object Oriented Systems. **IEEE Software**, p.136-146, sept. 2004.

WARREN, Ian; RANSOM, Jane. Renaissance: A Method to Support Software System Evolution. **IEEE Software**, p.415-420, 2002.