JOAQUIM ANTONIO PEREIRA

PROPOSTA DE UMA ESTRATÉGIA DE REENGENHARIA DE SISTEMAS LEGADOS PARA SISTEMAS ORIENTADOS A OBJETO

Monografia apresentada à Escola Politécnica da Universidade de São Paulo para conclusão do curso de Especialização em Engenharia de Software

Área de Concentração:
Reengenharia de Software

Orientador:
Prof. Dr. Jorge Luis Risco Becerra

São Paulo - 2003

AGRADECIMENTOS

Ao orientador Prof. Dr. Jorge Luis Risco Becerra pela valiosa orientação pelos conselhos, diretrizes e paciência

À minha família pelo incansável estímulo e compreensão.

SUMÁRIO

LISTA DE FIGURAS **5**

LISTA DE ABREVIATURAS E SIGLAS **6**

RESUMO **7**

ABSTRACT **8**

CAPÍTULO 1 INTRODUÇÃO **9**

1.1 CONSIDERAÇÕES INICIAIS 9
1.2 OBJETIVOS DO TRABALHO 10
1.3 ABRANGÊNCIA DO TRABALHO 11
1.4 MOTIVAÇÃO E JUSTIFICATIVA 11
1.5 METODOLOGIA 12
1.6 ESTRUTURA DA MONOGRAFIA 12

CAPÍTULO 2 SISTEMAS LEGADOS **14**

2.1 SISTEMAS LEGADOS 14
2.2 REENGENHARIA DE SISTEMAS 17
2.3 ENGENHARIA REVERSA DE SISTEMAS 18
2.4 FORWARD ENGINEERING 19
2.5 "CONVERTER" OU "MIGRAR" SISTEMAS LEGADOS 20
2.6 ELEMENTOS TEÓRICOS SOBRE OS SISTEMAS LEGADOS 21
2.6.1 Considerações sobre os sistemas legados 21
2.6.2 Categorias de sistemas legados 28
2.6.2.1 Aplicações altamente decompostas 29
2.6.2.2 Aplicações com dados decompostos 30
2.6.2.3 Aplicações com programas decompostos 31
2.6.2.4 aplicações não estruturada ou monolíticas 32
2.6.3 Fundamentos que propiciam a reengenharia de legados 33
2.6.3.1 Valor de negócio da aplicação legada 34
2.6.3.2 Requisitos para desenvolvimento e flexibilidade 34
2.6.4 Qualidade técnica de uma aplicação legada 34
2.6.5 Identificar estratégia para reengenharia de sistemas legados 36
2.6.6 Estratégia de reengenharia de sistemas legadas 37
2.6.6.1 Ignorar os sistemas legados 38
2.6.6.2 Rescrever 38
2.6.6.3 Integrar 39
2.6.6.4 Data warehouse 40
2.6.6.5 Migração Gradual 41

CAPÍTULO 3 MODELO PARA REENGENHRAIA DE SISTEMAS LEGADOS 43

3.1 ETAPAS PARA A REENGENHARIA DE SISTEMAS LEGADOS 44

3.2 PLANEJAMENTO PARA REENGENHARIA DE SISTEMAS LEGADOS 45

3.3 ANÁLISES E REQUISITOS DE REENGENHARIA DE SOFTWARE 51

3.3.1 Conservar a familiaridade entre os sistemas e usuários 51

3.3.2 Remodularizar o código legado com funcionalidades em duplicata. 52

3.3.3 Requerimentos funcionais 52

3.4 ARQUITETURA – AMBIENTE PARA REENGENHARIA DE SISTEMAS LEGADOS ... 52

3.4.1 Montagem do ambiente operacional 53

3.4.2 Análise de inventário 53

3.4.3 Reestruturação da documentação dos sistemas 54

3.4.4 Revisão do projeto de reengenharia de sistemas 54

3.4.5 Administrar esforço de migração – migrar, manter e controlar retrabalhos 54

3.4.6 Criar protótipos 55

3.5 REORGANIZAR O CÓDIGO LEGADO – PROJETO FÍSICO 56

3.5.1 Restruturação do código - primeira fase 57

3.5.2 Otimização das transformações 59

3.5.3 Código legado Reorganizado 59

3.6 CONVERSÃO – REENGENHARIA DE SISTEMAS – MIGRAR O CÓDIGO LEGADO64

3.6.1 Etapas para migrar código legado procedural para código orientado a objeto 65

3.6.2 Produtos gerados 66

3.6.3 Analisar o código legado 68

3.7 TESTES DA APLICAÇÃO MIGRADA 69

3.7.1 Testes Individuais 69

3.7.2 Testes Integrados 69

3.7.3 Validar e Homologar o sistema migrado - comparando resultados 69

3.8 IMPLANTAR EM PRODUÇÃO O SISTEMA LEGADO 70

CAPÍTULO 4 CONSIDERAÇÕES FINAIS 71

4.1 CONCLUSÕES 71

4.2 COMENTÁRIOS GERAIS 72

4.3 TRABALHOS FUTUROS 72

4.4 CONTRIBUIÇÃO 72

REFERÊNCIAS BIBLIOGRÁFICAS 73

LISTA DE FIGURAS

FIGURA 1 SISTEMAS LEGADOS 17

FIGURA 2 PROCESSO DE ENGENHARIA REVERSA 19

FIGURA 3 CATEGORIA DE APLICAÇÕES LEGADAS ALTAMENTE DECOMPOSTA 30

FIGURA 4 CATEGORIA DE APLICAÇÕES LEGADAS DECOMPOSTAS POR DADOS - DUAS CAMADAS 31

FIGURA 5 CATEGORIA DE APLICAÇÕES LEGADAS DECOMPOSTA POR PROGRAMA – DUAS CAMADAS..... 32

FIGURA 6 CATEGORIA DE APLICAÇÕES LEGADAS MONOLÍTICA - COMPONENTES EM UMA ÚNICA CAMADA 33

FIGURA 7 ESQUEMA PARA ANÁLISE DE ESTRATÉGIAS 36

FIGURA 8 MODELO DO PROCESSO DE REENGENHARIA DE SISTEMAS 45

FIGURA 9 RESTRUTURAÇÃO DE CÓDIGO 57

FIGURA 10 CONVERSÃO DE SISTEMAS LEGADOS 64

FIGURA 11TESTES NOS DOIS AMBIENTES E COMPARAÇÃO DE RESULTADOS .....**ERRO! INDICADOR NÃO DEFINIDO.**

LISTA DE ABREVIATURAS E SIGLAS

ADABAS	Banco de Dados - Software AG
API	Application Programming Interfaces
ATM	Asynchronous Transaction Managment
BATCH	Técnica de processamento em Lotes – Não Online
BMS	Base Mapping Suport
CICS	Customer Information Control System
COBOL	Linguagem de programação de terceira geração
DB2	Banco de Dados IBM
DBMS	Data Base Management System
DoD	Department of Defence
DOS	Disk Operating System
EJB	Enterprise JAVA Beans
GUI	Grafical User Interface
HW	Hardware
IMS/DC	Information Management System Data Communication
JAVA	Linguagem de Programação
JDBC	Java Data Base Connective
JSP	Java Serve Page
LAN	Local Area Network
MFS	Message Format Service
OO	Oriented Object
PC	Personal Computer
PF	Program Function
RECOVERY	Recuperação de arquivos
SEI	Software Enginnering Institute
SORT	Técnica para classificar arquivos
SYSOUT	Relatório em ambiente MAINFRAME IBM
SW	Software
TI	Tecnologia da Informação
VSAM RRDS	Virtual Storage Access Method Relative Record Data Set
VSAM KSDS	Virtual Storage Access Method Key Sequenced Data Set
WSAD	WebSphere Studio Application Developer

RESUMO

Este trabalho visa apresentar uma proposta para uma estratégia de migração de sistemas legados em tecnologias procedurais, para as novas tecnologias orientado a objetos, tornando mais rápido, seguro e transparente a absorção dessas tecnologias pelas organizações assim como propiciar economia de tempo e dinheiro em processos de migração. Como essa estratégia é de uso especializado e, pode ser aplicado a qualquer linguagem de programação, foi direcionado para o domínio da linguagem COBOL por meio da criação de algumas diretrizes. Com a engenharia reversa finalizada, o processo de reengenharia pode ser iniciado objetivando-se a disponibilização dos sistemas legados monolíticos em sistemas em três camadas e em ambiente baseado na WEB.

ABSTRACT

The aim of this monograph is to present one strategy of migration of legacy systems "oriented procedure" technologies, for new object oriented technologies, becoming faster, safer and more transparent the absorption of these technologies by organizations and to appease time and money economy in migration proccess. As this strategy is specialized use and, can be applied for any program language, it was specialized for the COBOL language dominion by mean of creation of some policy. With the reverse engineering, the process of reengineering can be started objectifying the availability of monolitics legacy systems in three layer systems in a WEB based environment.

CAPÍTULO 1 INTRODUÇÃO

Este capítulo visa apresentar as considerações iniciais, os objetivos, a abrangência, a motivação, a justificativa, a metodologia, a revisão da literatura consultada, bem como a estrutura desta monografia, dentro do contexto do tema "PROPOSTA DE UMA ESTRATÉGIA DE REENGENHARIA DE SISTEMAS LEGADOS PARA SISTEMAS ORIENTADOS A OBJETO".

1.1 CONSIDERAÇÕES INICIAIS

A grande maioria das aplicações de sistemas de informações existente atualmente, é aplicações velhas, desenvolvidos entre os anos 70 e meados dos anos 80 e mostram sinais de sua idade, e anos de remendos e consertos. Geralmente esses sistemas são vitais para a sobrevivência das organizações. Contudo está se tornando cada vez mais caro, a sua operação e manutenção. Alguns sistemas levam meses para simples melhorias [28].

As pressões de negócios para prover flexibilidade e informações oportunas e concisas para o gerenciamento de decisões e suporte operacional de crescimento, as aplicações legadas são inadequadas para esse fim.

Contudo, manter aplicações legadas está difícil para muitas organizações e não podem ser jogados fora; é difícil jogar fora aplicações que suportam serviços críticos tais como: Faturamento, Folha de Pagamento, Ativo Fixo, Contas a Pagar e Receber. Este é o dilema, e está cada vez mais difícil conviver com eles; também não se pode viver sem eles [28].

Desenvolver sistemas a partir **"do zero"** é na maioria das vezes, mais difícil do que quando existe algo pronto para ser usado como base. Em muitos sistemas legados pode-se aproveitar a estrutura geral de um programa existente, módulo para a montagem de menus, relatórios , consultas e a estrutura de operações de acessos e manutenção de data bases ou arquivos [21].

As organizações acumularam ao longo de anos de trabalho muito código de programas, que podem servir de base para a elaboração de novos sistemas. Contudo, desenvolver sistemas dessa forma pode induzir o desenvolvedor a erros ou problemas, tais como: aproveitamento de código sobre código, deixa o sistema ineficiente; sistemas difíceis de manter; descaracterização do sistema original, que talvez fosse mais prático partir para um desenvolvimento a partir do zero.

Além do aproveitamento do código deve-se aproveitar também as documentações de análise e de projeto, pois na construção de um sistema, sabe-se ser essa as etapas que mais consomem esforços para serem elaboradas e poderiam ser reutilizadas em novos desenvolvimentos. Essas documentações de análises e projetos devem ser sintetizadas em padrões , tanto de código, como análise ou mesmo de projeto.

Sistemas aplicativos dentro dos conceitos da tecnologia Orientado a Objetos tornam amigáveis as tarefas do dia a dia da Tecnologia da Informação, e tornam simples os processos de desenvolvimento e manutenções de sistemas, melhorando a qualidade dos programas; e como resultado final, obtém-se sistemas aplicativos estáveis e de melhor qualidade apresentando menores custos de manutenção.

1.2 OBJETIVOS DO TRABALHO

Este trabalho tem por objetivo principal apresentar uma "estratégia" de migração dos sistemas legados para sistemas orientados a objetos como uma alternativa de substituição de sistemas legados.

Sistemas orientados a objeto são de amplo domínio da comunidade da Tecnologia da Informação, e todos sabemos que sistemas orientados a objeto são sistemas que devido às suas características da tecnologia possuem uma certa facilidade à sua manutenção, possibilitando as corporações que utilizam essa tecnologia tornarem-se mais competitivas através dos sistemas de informação.

Redesenvolver todo o legado de sistemas numa corporação, é uma tarefa extremamente cara, demorada e que nem sempre é possível, pois as organizações têm problemas em manter os seus sistemas legados, pois a dinâmica a que estão submetidas impossibilitam as corporações de redesenvolver os seus sistemas legados, também pelo alto custo de desenvolvimento de novos sistemas.

Um outro objetivo deste trabalho é propiciar e incentivar a construção de ferramentas que possibilitem reescrever programas, a partir de códigos legados procedurais, transformando esses códigos em programas orientados a objeto, com grande produtividade e sem introduzir erros de regra de negócios, e essa tarefa não é uma atividade trivial.

1.3 ABRANGÊNCIA DO TRABALHO

A abrangência desta monografia estará contida no contexto da Reengenharia de software aplicativos de toda natureza, a ser aplicado em projetos para o re-aproveitamento de códigos legados nas corporações.

Um processo de reengenharia é igual a qualquer outro processo no qual as técnicas e padrões têm emergido em cada processo, aplicando-se a várias necessidades da reengenharia, objetivando custos benefícios mais atraentes nesses processos.

O produto final deste trabalho e apresentar uma estratégia para ser aplicada a qualquer linguagem de programação para incrementar e melhorar a qualidade nos processos de reengenharia, permitindo uma padronização nesses processos de migrar sistemas legados, reduzindo custos e riscos e incrementando a sua produtividade.

1.4 MOTIVAÇÃO E JUSTIFICATIVA

Esta monografia tem como motivação os seguintes objetos: uma proposta de uma estratégia de reengenharia de sistemas legados para sistemas orientados a objeto e a integração com ferramentas para reescrever sistemas legados.

O alto custo, prazos muitos longos incompatíveis com as necessidades das corporações para re-desenvolvimentos de Sistemas, pressões nos negócios, tornam a alternativa da reengenharia de sistemas legados uma alternativa atraente no que diz respeito a tempo e custo, justificando sobre maneira o desenvolvimento desse trabalho.

Outra motivação é que este trabalho ajudará as corporações a repensar em alternativa da reengenharia de sistemas legados como uma forma de atingir os seus objetivos na utilização dos seus sistemas aplicativos como suporte aos seus negócios, evitando altos custos em TI, para o desenvolvimento de sistemas a partir do "zero".

O alto custo de desenvolvimento de novos sistemas, as pressões de negócios, a Internet, e a globalização justificam plenamente a elaboração de um modelo de reengenharia de sistemas legados para sistemas orientados a objeto.

1.5 METODOLOGIA

No desenvolvimento dessa monografia foram adotadas as seguintes fases que nortearam a metodologia utilizada.

- Pesquisa: Nesta fase foram efetuados levantamentos e seleção da bibliografia utilizada nessa monografia e que foram consideradas relevantes. Foram consultados artigos pela Internet bem como obras de autores especializados no assunto, sendo muito intenso no início dos trabalhos.

- Análise do modelo de sistemas: Nesta etapa obteve-se o conhecimento dos modelos de sistemas para a definição das estratégias a serem utilizadas nos processos de reengenharia de sistemas legados.

- Modelagem para a reengenharia de legados: Fase em que a reengenharia dos sistemas legados será aplicada para o a geração de sistemas orientados a objetos. Todos os conhecimentos dos sistemas, extraídos na fase anterior irão alimentar todo o processo para a reengenharia desses sistemas legados.

- Transformação dos legados: Etapa em que o código legado procedural será transformado em código orientado a objetos, sem que as funcionalidades e objetivos dos programas sejam afetados.

- Elaboração da Monografia: Consistiram de pesquisas em livros sobre sistemas legados, artigos encontrados na internet sobre o tema em questão e por experimentos na área de reengenharia de sistemas legados, bem como a valiosa orientação do Prof. Dr. Jorge Becerra.

1.6 ESTRUTURA DA MONOGRAFIA

Esta monografia está organizada como segue.

- Primeiro Capítulo: São apresentados as considerações iniciais, os objetivos, a abrangência, a motivação, a justificativa e o método utilizado para a elaboração deste trabalho.

- Segundo Capítulo: É abordada a base teórica com o objetivo do entendimento dos conceitos de Sistemas legados. Também serão apresentados os conceitos de migração de sistemas.

- Terceiro Capítulo: É apresentada uma proposta de uma estratégia de reengenharia de sistemas legados para sistemas orientados a objetos, como um modelo de estudo de caso.

- Quarto Capítulo: Serão apresentadas as conclusões, os comentários, trabalhos futuros e em que esta monografia pode contribuir.

CAPÍTULO 2 SISTEMAS LEGADOS

Neste capítulo serão discutidas as teorias que serviram de base para o entendimento de sistemas legados nas corporações e a convergência para a Reengenharia desses Sistemas Legados, para Sistemas Orientados a Objeto.

No item 2.1 será abordado, **o que é sistema legado**, tanto do ponto de vista da lingüística como do ponto de vista conceitual da tecnologia da informação, bem como conceitos de reengenharia de sistemas, de engenharia reversa de sistemas, e de engenharia avante.

No item 2.2 serão abordados os elementos sobre sistemas legados, envolvendo considerações sobre sistemas legados, paradigmas entre outros, sobre linguagem Cobol, definida como uma linguagem obsoleta, a manutenibilidade de software, as categorias e tipos de sistemas legados nas organizações a partir de uma perspectiva de reengenharia de aplicações e como elas são classificadas. Serão abordados também e fundamentos .que propiciam a reengenharia de sistemas legados.

No item 2.3 será abordados sucintamente os conceitos e princípios de padrões (**Pattern**) para processos de reengenharia de sistemas legados, para sistemas orientados a objetos.

- **Algumas definições conceituais**

Neste capítulo são apresentadas algumas definições conceituais sobre termos utilizados para definir o que é um sistema legado, o que é reengenharia de sistemas, o que é engenharia reversa de sistemas, e o que é engenharia avante, bem como as diferenças entre migrar e converter.

2.1 SISTEMAS LEGADOS

A discussão conceitual do que seja um sistema legado é ampla, mas de acordo com o dicionário [Aurélio] , na língua Portuguesa o termo legado significa "objeto que alguém deixa a outrem, aquilo que alguém transmite a outrem , aquilo que uma geração, escola literária, etc, transmite à posteridade".

Ainda no campo lingüístico na língua inglesa, segundo o [Webster Dictionary] , "legado é alguma coisa de valor recebido de um ancestral ou de um predecessor ou do passado". O

termo legado pode ser e está sendo usado em muitos contextos, tais como: Legados LANs, tratam de legados de redes Ethernet LANs ; Legados sistemas operacionais, tratam do PC DOS e Legados de estilos de gerenciamento, tendo-se como exemplo o Taylorismo, que foi o movimento de gerenciamento cientifico, defendido por Frederyck Taylor [28].

A expressão legado é também usada para indicar que alguma coisa não está na moda, tomando-se como por exemplo, um sistema que não esteja construído na linguagem de programação JAVA, embora tenha sido construído o mais recente possível, seja tratado como um sistema legado; uma aplicação legada, é uma aplicação que tem o seu valor herdado do passado [28].

Aplicações legadas também são definidas como "grande sistemas de software que não sabemos como conviver sem eles, e são vitais para as organizações " [3].

" Qualquer sistema de informação que resista significamente a modificações e evoluções para atender novas e sucessivas mudanças de código para atender aos requerimentos das regras dos negócios" ". [5]".

Existem autores que são bem específicos para definir um sistema legado: "qualquer aplicação que esteja em produção". [24], isto é, todo e qualquer sistema quer ele esteja construído na mais avançada tecnologia ou não.

"Pode-se definir aplicações legadas como qualquer software disponibilizado na produção independentemente da plataforma que ele esteja sendo utilizado, da linguagem em que ele foi escrito ou do tempo que ele tenha sido posto em produção" [27].

Segundo o Forrester Research Survey conduzido pela IBM em 1993, 80% dos Bancos de Dados das corporações estavam armazenados em IMS/DB, gerenciador de Banco de Dados da IBM, projetado no início dos anos 60.

Especificamente, aplicações ou sistemas legados são classes de aplicações que são segundo [4], [3], [11] e [16]:

- Crucial no dia-a-dia das corporações;
- Houve muito investimento durante muitos anos e que não podem simplesmente ser jogado fora;
- Mais de 200 Bilhões de linhas de código e milhões de programas;

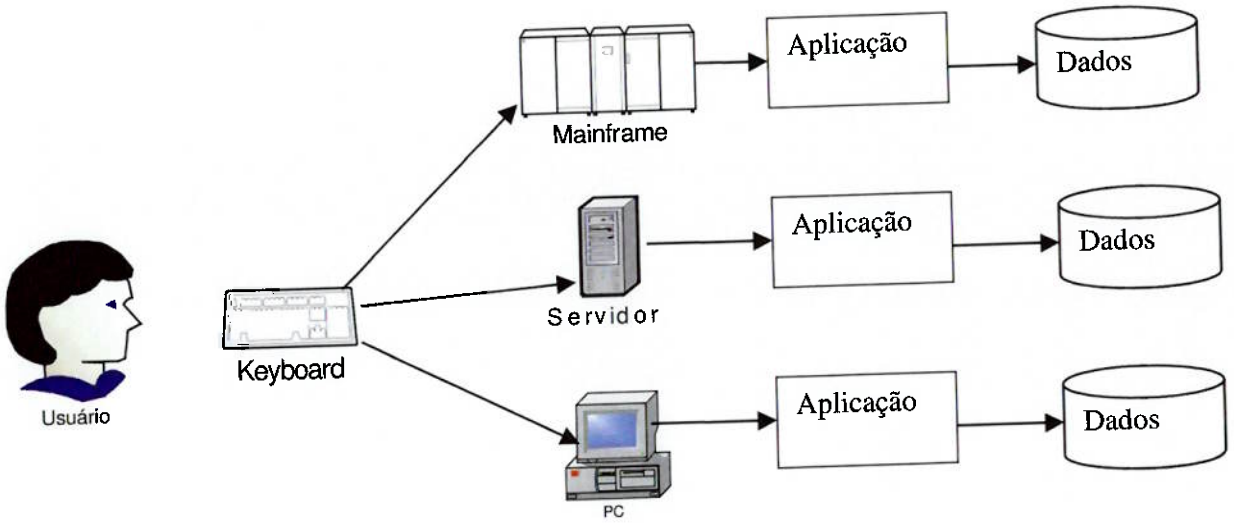
- Usados diariamente em muitos milhões de transações ATMs, Batch ou on-line;
- Não estão bem documentadas e dificultam o seu entendimento para manutenções corretivas e evolutivas;
- Inflexível, dispendioso, consumindo muito tempo e riscos para manutenção e mudanças;
- Está baseado em tecnologias "velhas" de Bancos de Dados e arquivos indexados ou seqüenciais;
- Escritos em linguagens de programação também "velhas";
- Interface de usuários com telas no formato texto, ao invés de GUI
- Integradas verticalmente e construídas de forma monolítica;
- São repositórios de anos de experiências e práticas de negócios das corporações, envolvendo muitas regras de negócios embutidas nos códigos legados.

Nota-se que através dos anos, os sistemas legados incorporaram substanciais conhecimentos das corporações, envolvendo necessidades de mudanças, desenho da aplicação e regras de negócios.

Contudo, o conhecimento das regras de negócios e decisões técnicas muitas vezes não estão documentadas e se encontram embutidas nos códigos dos sistemas legados. Tais conhecimentos são difíceis de serem recuperados depois de anos e décadas de operações destes sistemas bem como pelas mudanças do pessoal envolvidos na sua construção. Grande parte do código dos sistemas legados foi desenvolvido em linguagens (hoje visto) arcaicas, escritas entre 10-25 anos atrás e provavelmente sem metodologias de desenvolvimento, tornando a sua manutenibilidade difícil e cara. O resultado disso são sistemas legados que apresentam deficiências e custos elevados para o atendimento de mudanças da demanda, geradas pelas necessidades impostas ao sistema, seja ela evolutiva, corretiva ou em função de novas tecnologias a serem implementadas nas corporações.

Na visão de um usuário de um de sistema, qualquer aplicação, esteja ela numa plataforma cliente/servidor, na plataforma Mainframe ou em uma plataforma PC ,esse Sistema ao entrar em produção, seja considerada um sistema legado. A figura abaixo sintetiza esses conceitos de Sistemas Legados.

Figura 1 sistemas legados



2.2 REENGENHARIA DE SISTEMAS

A reengenharia tem por finalidade examinar e alterar um sistema existente para reconstruí-lo em uma nova forma e depois implementá-lo em uma nova forma [9]. A reengenharia tem como objetivo principal melhorar a qualidade global do sistema, mantendo, em geral, as funções do sistema existente. Mas ao mesmo tempo, pode-se adicionar novas funções e melhorar o desempenho [20]. Segundo Jacobson [13], a reengenharia consiste da engenharia reversa , seguida de mudanças no sistema (que podem ser mudanças de funcionalidade ou mudanças de técnica e de implementação) e seguida da engenharia forward. Ou seja, "reengenharia é o processo de criar uma descrição abstrata do sistema, elaborar mudanças em alto nível de abstração e então implementá-la no sistema" [21]. De acordo com Sneed [23], a reengenharia engloba a reengenharia do procedimento empresarial, a reengenharia dos dados a reengenharia do software, e a reciclagem (que produz componentes reusáveis, de maneira análoga ao processo de retirada de peças aproveitáveis de um automóvel abandonado).

Uma das grandes motivações para a aplicação da reengenharia de sistemas legados é a diminuição dos altos custos de manutenção de sistemas que se devem a diversos fatores [28].

A manutenção contínua faz com que a implementação torne-se inconsistente com o projeto original, o código torne-se difícil de entender e sujeito a muitos erros. Além da documentação desatualizada. As linguagens de programação em que esses sistemas legados foram implementados estão ultrapassadas, não havendo suporte por parte dos seus fabricantes e o ensino destas linguagens pelas universidades contribuindo dessa forma, para a escassez de profissionais que as dominem.

Assim a reengenharia torna-se uma área interessante para ser explorada dentro do campo da engenharia de software, que objetiva melhorar a produtividade e qualidade dos processos de "modernização" de software. Para cumprir tal objetivo a reengenharia de software precisa entender corretamente os programas existentes, a partir dos códigos fontes e documentação disponíveis, para facilitar a realização de mudanças e reconstrução do software numa nova tecnologia. Portanto, o propósito da reengenharia de software é facilitar a execução de mudanças e correções, a recuperação do desenho em um nível mais alto de abstração, o redesenho e a reprogramação de um software.

2.3 ENGENHARIA REVERSA DE SISTEMAS

A engenharia reversa produz uma imagem de "orifício mágico". Alimentamos uma listagem fonte, desestruturada, não documentada, no orifício e no outro extremo sai uma documentação completa para o programa de computador. Infelizmente o orifício mágico não existe [20]

Engenharia Reversa normalmente é empreendida com o objetivo de reprojetar um sistema para melhorar a sua manutenção ou produzir uma cópia de um sistema sem acesso às informações de seu projeto original.

A engenharia reversa é uma área de interesse emergente principalmente pelo volume de sistemas legados que precisam evoluir. Diz-se Engenharia Reversa porque, ao contrário da engenharia tradicional, partimos do produto para a sua definição. Portanto, antes de evoluir um determinado sistema de software, faz-se necessário reverte-lo a descrições mais abstratas com o objetivo de facilitar a compreensão do que o sistema é, como ele funciona e como não funciona, para só então modificá-lo. Como se trata de um processo e não de um ferramental, a Engenharia Reversa pode ser aplicada a cada um dos três aspectos principais de um sistema: dados, processo e controle.

A seguir é apresentado na figura 2 o fluxo do processo de engenharia reversa, que através do código fonte original, extraídos todos os processos de um programa monolítico e já separados em camadas.

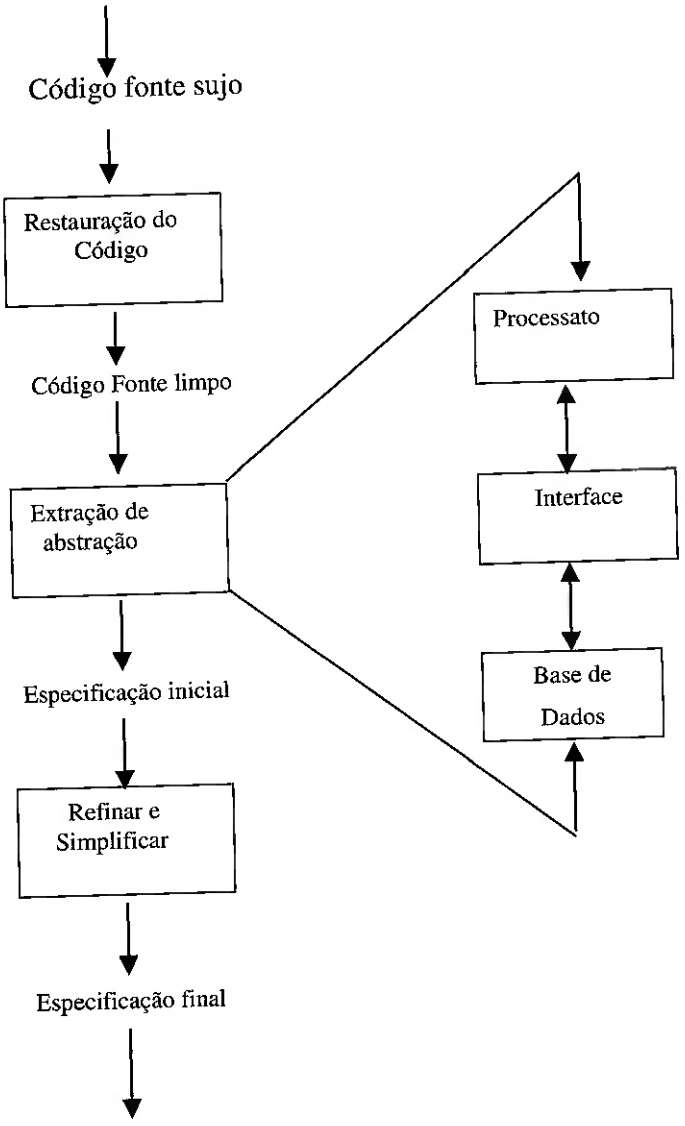


Figura 2 Processo de Engenharia Reversa

2.4 ENGENHARIA AVANTE (FORWARD ENGINEERING)

É também chamada de renovação. Não apenas recupera as informações do software existente mas o altera e o reconstitui num esforço para aprimorar sua qualidade. O software que passa

por reengenharia implementa funções do sistema existente e também adiciona e/ou melhora a sua performance nas funções que já existiam e que foram mantidas.

2.5 "CONVERTER" OU "MIGRAR" SISTEMAS LEGADOS

Será essa é uma questão de semântica, ou existe diferenças entre o que é migrar para o que é converter? Profissionais de TI, normalmente usam os dois termos quando se referem a um processo de **"reengenharia"** de sistemas. Segundo o dicionário [AURÉLIO] na língua portuguesa, a palavra **"converter"** ou na língua inglesa a palavra **"conversion"** semanticamente essas duas palavras tem o mesmo significado – "fazer mudar para, transformar algo em alguma coisa" ". Já a palavra "migrar" na língua portuguesa ou "migrate" na língua inglesa, semanticamente também tem o mesmo significado – "mudar de lugar". Assim, é possível deduzir que semanticamente essas palavras não tem nada a ver uma com a outra, mas no entanto, erradamente, são usadas quando alguém expressa o desejo de fazer reengenharia em sistemas legados.

Apesar de terem significados diferentes as duas palavras são amplamente utilizadas para expressar as mesmas necessidades de mudanças de sistemas, ou processo de mudanças, como por exemplo, conversão do bug do milênio ou migrar os sistemas legados procedurais para sistemas orientado a objeto. O que se percebe é que "converter" é amplamente utilizado em projetos de menor impacto e "migrar" em projetos de maior impacto.

Migrar ou converter remetem a um processo de reengenharia de sistemas. Os autores constantes na bibliografia, cujas obras forneceram os subsídios teóricos que serviram de base para essa dissertação, utilizam a palavra "reengenharia" para expressar mudanças num software. Assim, exploraremos alguns conceitos de vários autores do que vem a ser a "reengenharia" que têm por finalidade examinar e alterar um sistema existente para reconstruí-lo em uma nova forma e depois implementá-lo em uma nova forma [9].

A reengenharia tem como objetivo principal melhorar a qualidade global do sistema, mantendo em geral as funções do sistema existente. Mas, ao mesmo tempo, pode-se adicionar novas funções e melhorar o desempenho [20] . Segundo Jacbson [13], a reengenharia consiste da engenharia reversa, seguida de mudanças no sistema e seguida da engenharia avante. Assim a "reengenharia" é o processo de criar uma descrição abstrata do sistema, elaborar mudanças em alto nível de abstração e então implementá-las no sistema. A

"reengenharia" engloba a reengenharia do procedimento empresarial, a reengenharia dos dados, a reengenharia do software, e a reciclagem (que produz componentes reusáveis de maneira análoga ao processo de retirada de peças aproveitáveis de um automóvel abandonado).

Uma das grandes motivações para a aplicação da reengenharia é a diminuição dos altos custos de manutenção de sistemas, que se devem a diversos fatores [28]. As manutenções constantes descaracterizam os sistemas sobre maneira tornando inconsistente com o projeto original, código fontes difíceis de serem entendidos, mantidos e sujeitos a erros.

2.6 ELEMENTOS TEÓRICOS SOBRE OS SISTEMAS LEGADOS

Apresentar os elementos que compõe uma base teórica dessa dissertação sobre os sistemas legados, é o principal objetivo desse capítulo. O entendimento sobre paradigmas que envolvem sistemas legados é apresentado para servir de base ao capítulo 3.

A seguir, são apresentados conceitos necessários para elaborar uma proposta para uma estratégia de reengenharia de sistemas legados.

2.6.1 Considerações sobre os sistemas legados

As diferentes arquiteturas dos sistemas legados onde esses sistemas foram desenvolvidos, nas mais diversas tecnologias tanto de Hardware como de Software e nas mais diversas metodologias de desenvolvimento de sistemas, podem dar uma idéia de uma "torre de babel" na área de TI, prevalecendo uma idéia de descontrole dos profissionais de TI. Antes de detalhar as várias arquiteturas de sistemas legados encontrados, é importante salientar as falsas idéias sobre sistemas legados.

Desinformações sempre circundam os debates a respeito desse tema **"sistema legado"**. Elas são originárias de várias fontes, muitas das quais têm pouca experiência prática de trabalhos com estes tipos de sistemas ou podem ter planos de novo desenvolvimento ou compra de soluções prontas no mercado (pacotes) que limitam a importância do papel dos sistemas legados dentro da empresa [27].

Com o advento das novas tecnologias, em especial: da internet, de sistemas orientados a objeto e de aplicações cliente/servidor, entre outras, tornam qualquer sistema legado alvo de "discriminação tecnológica' dentro da organização, interferindo por vezes na evolução desses sistemas".

Aceitar estes mitos reduz perigosamente a capacidade de uma organização em fornecer soluções de negócios onde o tempo é fator critico. Subestimar o valor das arquiteturas de negócios e suas complexidades resultará em uma demora ou falha de uma grande variedade de iniciativas de negócios. Reconhecer esses desafios, por outro lado, é o primeiro passo para a criação de uma estratégia de transformação da arquitetura legada, que fornecerá soluções de negócios críticos, nos tempos ou prazos requeridos pelos usuários [27].

A seguir, são apresentadas as considerações que envolvem os sistemas legados nas corporações [27].

- **Aplicações legadas agregam poucos valores aos negócios**

Aplicações legadas são o sangue da vida das empresas, porque elas processam no dia a dia os dados críticos dessas corporações. A maioria dos sistemas legados estão escritos em linguagens tidas como de terceira geração. Segundo Stephanie Wilkinson, "estas aplicações escritas em sua grande maioria em COBOL processam até 85% de todo os negócios global" [26] . Se estas aplicações repentinamente desaparecerem como num passo de mágica, seria uma catástrofe para as empresas. Reconhecer os sistemas legados como importante ativo da empresa é o primeiro passo na criação de uma estratégia que trate os desafios relacionados com esses sistemas legados. Sem dúvida nenhuma, esses legados agregam valores aos negócios nas empresas, pois sem esses legados as organizações não teriam evoluído em seus negócios [27].

Também se percebe uma falsa idéia de que sistemas "velhos" não agregam valores aos negócios das empresas por comparar-se, precipitadamente ou sem nenhum critério, esses sistemas legados com sistemas voltados a tecnologias WEB, como só os sistemas voltados à internet agregassem valores aos negócios das corporações .

- **Sistemas em web substituem rapidamente sistemas legados**

Sistemas baseados em WEB estão com alguma rapidez substituindo os sistemas legados nas organizações. Os front ends baseados em WEB podem parecer estar substituindo aplicações legadas, mas o volume de transações on-line de sistemas legados continua crescendo. A estrutura de processamento mais usada nestas transações é o CICS (Customer Information Control System) da IBM que alcançou a marca de 20 bilhões de transações processadas por dia, dados de 1998. Isto era mais que o total de acessos por dia da internet nesse mesmo período. Se os sistemas legados estivessem sendo substituídos com parece, poderia se esperar que houvesse uma redução de transações on-line, sob CICS, o que de fato não aconteceu. Prova disso é que por volta do ano 2.000 a IBM anunciou que o número de transações CICS que eram processadas diariamente, ultrapassava os 30 bilhões de transações, um acréscimo de 50% sobre a quantidade processamento desse tipo de transações no ano de 1998.

O número de clientes que passaram a utilizar o CICS também cresceu, de 30.000 licenças CICS em 1.993, para 50.000 licenças em 1.999. Todos estes dados acima mencionados foram publicados no artigo "The Evolution of CICS: 30 Years Old and Stil Modern" por T. Scott Ankrum em The Cobol Report. Nota-se com clareza que o CICS continua sendo utilizado como suporte principal da maioria dos sistemas comerciais para o processamento de suas transações on-line. E continua a crescer. [27].

- **Cobol é uma linguagem obsoleta**

O Cobol é uma linguagem não mais usual, está obsoleta e não tem sido melhorada ou evoluída. Universidades Americanas tiraram o COBOL de seus currículos, por considerarem ser uma linguagem obsoleta. Ao contrário, existem no mundo mais de 200 bilhões de linhas de código Cobol, representando mais de 60% dos códigos de software existentes no mundo [27].

Segundo Stephanie Wilkinson, em seu artigo referenciado no item 2.2.2 as aplicações Cobol tem também uma expectativa de crescimento de 5 bilhões de linhas de código anualmente e 15% de todas novas aplicações que funcionarão nos próximos 5 anos serão escritos em Cobol [27].

Contudo o Cobol evoluiu, permitindo criar programas orientados a objetos e voltados para a Web. Também já há algumas Universidades Americanas que estão mantendo em seus

currículos os cursos para formar programadores Cobol, porque há ainda muito espaço para essa linguagem [27].

Sob o prisma dessas informações, como pode então muitos profissionais da tecnologia da informação afirmarem que o Cobol é uma linguagem morta? Só a falta de informações podem permitir tal engano. No Brasil, nota-se, que em grandes centros de tecnologia da informação o Cobol será ainda por muitos anos utilizado, até porque os negócios dessas corporações, do tipo missão crítica, ainda são confiadas o seu desenvolvimento na linguagem Cobol, em detrimento de outras linguagens.

- **Dados legados podem ser acessados pela web**

Dados de sistemas legados armazenados podem ser aproveitados para serem acessados por aplicações disponibilizadas na WEB. Esses dados legados por vezes estão definidos de forma redundante e inconsistente. As estruturas de dados de sistemas legados devem ser avaliados com relação a sua consistência pois, no mesmo dado, tendo-se como exemplo um número de cliente ou telefone podem significar diferentes coisas para diferentes unidades de negócios. Estas atividades juntam-se ao fato de que algum destes dados contém valores inválidos ou inconsistentes. E nesse processo, de criar diferentes interfaces para acessar estes dados através da WEB, pode ser uma proposição arriscada, com resultados que possam afetar a credibilidade da Web na empresa [27].

- **A funcionalidade dos sistemas legados já não atende mais**

A funcionalidade de sistemas legado não são mais válido. Este é provavelmente o maior e o dos mais falsos conceitos sobre sistemas legados. Estas funcionalidades podem ser difíceis para entender a sua engenharia, ser também difíceis de se integrar (fora de um ciclo normal de processo) a outros sistemas legados, ou ter sido definido redundantemente, muito embora o código e a regra do negócio do legado funcione muito bem e seja seguro. Nas empresas, a maioria dos sistemas legados contém regras de negócios que não podem ser modificados dinamicamente para atender aos novos requerimentos que atenda a evolução de novas funções ou de negócios, das empresas [27].

Então, porque esses sistemas ainda continuam sendo utilizados nas empresas? Por quê não foram descartados ou substituídos por soluções de software prontas "pacotes" ou novos desenvolvimentos?

- **Adaptar sistemas legados atende novos negócios na web**

Adaptar sistemas legados não atende novos negócios na WEB, pois a fragmentação funcional dos sistemas legados que circunstancialmente são causadas pela evolução hierárquica e desordenada das estruturas dos sistemas e dados legados, criaram uma situação onde os sistemas legados cumprem as suas tarefas gradativamente. Cada sistema executa uma etapa de um processo maior e outras etapas subseqüentes são confiados a outros sistemas para continuar o ciclo ou o processo. São sistemas com forte interação entre si.

Esta abordagem está em conflito direto com as necessidades de sistemas no ambiente e-business que acionam regras de negócio e acesso a dados sob demanda. Sistemas legados adaptados para WEB podem atender requisitos de curto prazo sendo, pois uma atividade legitima de desenvolvimento de aplicações. Porém, os requisitos de longo prazo de funções chave neste ambiente não serão atendidas dentro da empresa. Um plano de transformação mais amplo é necessário para alcançar este objetivo [27].

- **Novos sistemas desenvolvidos podem ignorar sistemas legados**

Não se pode ignorar os sistemas legados. Estudos (notadamente feito por Gane e Sarson) têm demonstrado que novos sistemas **"herdam"** 80% ou mais da funcionalidade dos sistemas existentes. Mesmo que fosse entre 40% e 50%, das regras de negócios que existem e estão embutidos num código de um sistema legado, mesmo assim, tenderia a dificultar a reprodução em qualquer grau de exatidão das funcionalidades de um sistema legado. Qualquer esforço de reconstruir ou substituir um sistema legado, em todo ou em parte, deve-se ter total domínio das regras de negócios para serem colocados no novo sistema que está sendo reposto ou reconstruído. O entendimento de um sistema legado é o requerimento mínimo para determinar que porção desse sistema legado precisa ser reconstruído [27].

➢ **Manutenibilidade de software**

A manutenibilidade pode ser definida qualitativamente como a facilidade com que um software pode ser entendido, corrigido e adaptado. A manutenibilidade é a meta primordial que orienta os passos de um processo de engenharia de um software, seja para a manutenção ou a evolução de software, que serão discutidos a seguir.

Modificar um software é uma atividade trivial no dia a dia das organizações, seja ela para correção de um "bug" ou para atender a uma necessidade legal e essa simples atividade exige muito cuidado. Infelizmente, toda vez que uma modificação de software e introduzida num procedimento lógico complexo, o potencial de erros tende a crescer. Na correção de um erro, muitas vezes o desenvolvedor não consegue prever as implicações dessa correção em outros módulos do software, provocando efeitos colaterais na aplicação. Quanto mais manutenção o software sofra maior será a sua descaracterização frente ao projeto original.

A reengenharia é um recurso importante para se recuperar as informações de projeto de um software existente, que permita atender a evolução do software legado, que possibilite as organizações atualizações tecnológicas, envolvendo a implementação de novos requisitos de usuário para atender as suas necessidades, adequar os software às necessidades de regras de negócio, bem como adequar os seus sistemas para as novas tecnologias de hardware e do software, que a cada instantes são disponibilizados para o mercado. Esta é provavelmente a forma mais rápida e menos onerosa do ponto de vista econômico para ajudar as organizações na evolução dos seus sistemas de informação.

➢ **Manutenção de software**

"Há 30 anos a manutenção de software foi caracterizada [7] como um "iceberg". No início dos anos 70, o "iceberg" de manutenção era suficientemente grande para afundar um "porta-aviões. Hoje, poderia afundar toda a Marinha" [20].

A manutenção de software existente pode ser responsável por mais de 60% de todo o esforço despendido por uma organização de desenvolvimento e a porcentagem continua a crescer ã medida que mais software é produzido [12]. Isso poderia levar a questionamentos, por que é necessária tanta manutenção e por que é despendido tanto esforço para a manutenção de sistemas. [20]. Osborne e Chikofsky [19] dão uma resposta sobre esse assunto:

"Grande parte do software do qual as organizações dependem atualmente, tem em média 15 anos. Mesmo quando esses programas foram criados as melhores técnicas de projeto e codificação conhecidas na época, o tamanho do programa e de armazenamento eram preocupações importantes. Depois migraram para novas plataformas ajustando para nova tecnologia de hardware e software e foram melhoradas para melhor atender as necessidades do usuário, isto tudo sem uma preocupação com a arquitetura global.

O resultado foi estruturas mal projetadas, codificadas e documentadas e de lógica pobre."

A natureza onipresente da modificação permeia todo o trabalho de construção de software. Modificações são inevitáveis quando sistemas baseados em computador são construídos. Conseqüentemente, é necessário desenvolver mecanismos para avaliar, controlar e realizar modificações.

Manutenção de software é, sem duvida, muito mais do que "concertar erros". Pode-se definir manutenção descrevendo quatro atividades [25], que são desenvolvidas, depois de um programa ser liberado para uso. Quais sejam: manutenção corretiva, manutenção adaptativa, manutenção perfectiva ou de melhoria e manutenção preventiva ou reengenharia. Para IAN em "Software Engineering" considera apenas três tipos de manutenção: A corretiva a adaptativa, a perfectiva. Apenas cerca de 20% de todo o trabalho de manutenção é gasto "consertando erros". Os restantes 80% são gastos adaptando sistemas existentes efetuando modificações no seu ambiente externo, fazendo melhorias solicitadas por usuários e submetendo uma aplicação a reengenharia, para uso futuro. Quando a manutenção abrange toda essa atividades, é realmente fácil ver porque absorve tanto esforço[20].

- **Preservar os investimentos e usar novas tecnologias**

Nos últimos anos, a frase – a Internet muda tudo – é uma das novas leis de TI. E é verdade, mesmo depois das quedas dos chamados **"ponto com"** a Internet é um fator estratégico no planejamento das gerencias de TI.

Mas, uma coisa não se modificou: a informação continua a ser um fator fundamental para o crescimento de uma organização, as regras de negócio não mudaram ou pouco mudaram.

Os novos desafios e possibilidades que nasciam com as tecnologias antes da Internet, e de sistemas orientados a objetos, incluem em si um problema para muitas áreas de TI. Como

tornar possível o uso das novas tecnologias, sem perder os investimentos feitos nos sistemas legados. Este é o maior desafio dos profissionais de TI.

Os sistemas, desenvolvidos aos longos dos anos de informatização nas organizações, contam com informações valorosas que rodam com estabilidade e confiabilidade. Por outro lado, estes sistemas encontram-se em arquiteturas fechadas e inflexíveis já explorados neste trabalho.

Por volta de 85 % das aplicações, ainda mantém as suas informações sobre processos e parceiros dentro de uma tecnologia obsoletas, principalmente escritas na linguagem COBOL [27]. Milhares de "homens-ano" de trabalho foram investidos em desenvolvimento e manutenção destas aplicações.

Certamente não fazer nada, não é a melhor opção a ser tomada. Um bom exemplo que define estes conflitos de objetivos é o setor financeiro, onde na economia global é o setor que mais investe na tecnologia da informação. Um banco que não oferece novos serviços como "internet banking", está fora da realidade tecnológica. Por outro lado, o núcleo dos negócios, as regras de negócios continuam a ser as mesmas, elas não mudaram com a evolução da Internet. Bancos ainda fazem aplicações, efetuam empréstimos, financiam a produção e estão dando créditos aos clientes, e obviamente calculam as taxas de juros e controlam os seus recebimentos. Fizeram isso antes da era de informatização e estarão fazendo assim por muitos e muitos anos e não importa o quanto a tecnologia possa evoluir.

Assim, preservar os investimentos é sem dúvida muito importante para as organizações, e a forma que pode ajudar neste objetivo de preservar investimentos de décadas de desenvolvimento de sistemas é a reengenharia de software para recriar sistemas legados para adequar esses sistemas as novas tecnologias disponíveis e agregar valores aos negócios das corporações.

2.6.2 Categorias de sistemas legados

Em uma organização, as aplicações legados consistem de dados legados, da sua funcionalidade (código da regra de negócio, a lógica da aplicação) e de interface de usuário APIs). A partir de uma perspectiva de reengenharia de uma aplicação, as aplicações legadas

podem ser classificadas nas categorias descritas a seguir e ilustradas nas figuras 3 4, 5 e 6 abaixo [4] e [5] .

- **Aplicações altamente decompostas**
- **Aplicações decompostas por dados**
- **Aplicações decompostas por programas**
- **Aplicações monolíticas**

2.6.2.1 Aplicações altamente decompostas

São aplicações bem estruturadas e bem construídas, apresentando as seguintes características: Primeiro, todos os componentes da aplicação são separadas em interfaces de usuário (as APIs), no código da aplicação (regras de negócio) e em serviços de acessos aos dados. São aplicações já construídas em três camadas. As interfaces dos serviços de acesso aos dados são muito bem definido permitindo que os dados da aplicação possam ser acessados remotamente sem a necessidade de se chamar funcionalidades de aplicação. Segundo, os módulos da aplicação são independentes entre si, sem a interdependência hierárquica entre os programas. Finalmente os módulos da aplicação tem de ser bem definidos, entre as interfaces com os serviços aos bancos de dados, interfaces com usuário e com outras aplicações.

São também consideradas aplicações altamente decompostas quando elas são desenvolvidas usando técnicas de **design** estruturadas ou as técnicas de orientação a objeto onde cada objeto tem a sua própria interface. Essas arquiteturas de aplicações são as mais amigáveis para a transição e acesso externo e permitem que as aplicações legadas possam ser acessadas diretamente e os seus componentes possam ser substituídos gradualmente, durante um processo transição. Essa arquitetura de aplicações possuem alta capacidade de manutenibilidade dos seus componentes [28].

A figura a seguir ilustra a categoria de programas de sistemas legados altamente decompostas, em que os objetos estão separados dentro do programa.

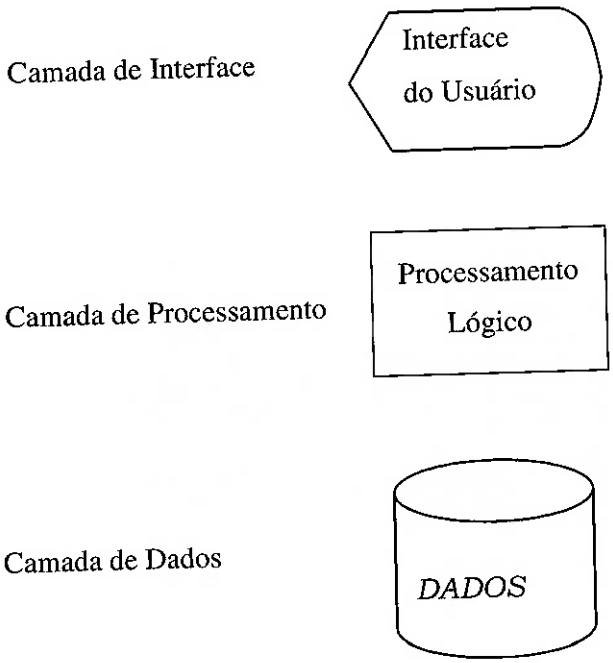


Figura 3 Categoria de Aplicações Legadas Altamente Decomposta

2.6.2.2 Aplicações com dados decompostos

São aplicações semi-decompostas por componentes de dados e de interface de usuário e lógica em que os componentes estão claramente em duas camadas cujas características são apresentadas a seguir::

Primeira característica, os componentes da aplicação são separados em duas unidades: Serviços de acesso aos dados e uma mistura de processamento de interface de usuários e processamento de lógica da aplicação, numa só unidade lógica. Têm-se assim duas camadas lógicas de aplicações onde os dados estão em uma camada e a interface de usuário e a lógica dos programas estão em outra camada.

A segunda característica, as interfaces para outras aplicações devem estar bem definidas. Nesta categoria de aplicações os dados podem ser acessados diretamente, mas a lógica não. Neste tipo de arquitetura os serviços de acesso aos dados podem ser reestruturados, bem como acessados remotamente através de programas (a segunda camada) ou através de ferramentas, em função das interfaces de serviços de dados. Idealmente as interfaces dos serviços de acessos aos dados podem ser projetados de tal forma, que possa permitir a substituição de gerenciadores DBMS (Data Base Management System) por outro DBMS,

como por exemplo: trocar Informix por Oracle ou Adabas por DB2, sem que isso possa impactar as aplicações. Infelizmente a lógica das aplicações dessa categoria de aplicações legada está vinculadas com o processamento da interface do usuário e deste modo não pode ser chamada remotamente. Mas essas aplicações, mesmo assim ainda são amigáveis e com alguma semelhança do ponto de vista de reengenharia da aplicação com as aplicações altamente decompostas, definidas no item 2.2.1 [28].

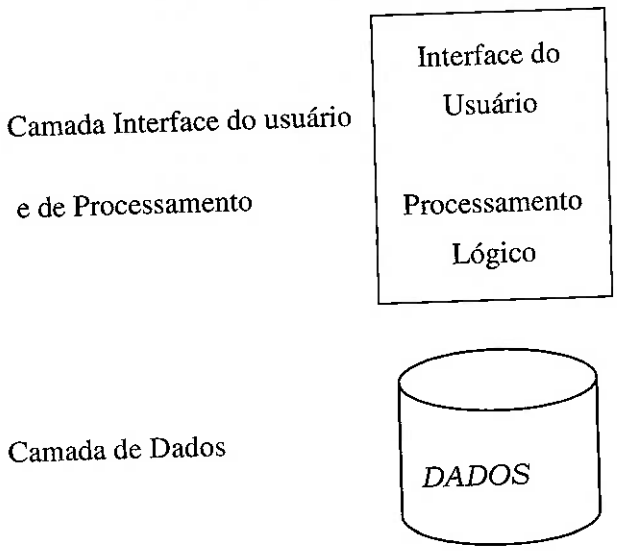


Figura 4 Categoria de Aplicações Legadas Decompostas por Dados - Duas camadas

2.6.2.3 Aplicações com programas decompostos

São aplicações semi-decompostas por componentes de interface de usuário e de lógica e acesso aos dados em que os componentes estão e em duas camadas e que são apresentadas a seguir::

Aplicações legadas com programas decompostos são também semidecompostas distinguindo-se das características apresentadas no item 2.6.2.2, nos seguintes pontos: Os componentes da aplicação estão separados em duas unidades, a unidade de processamento da interface do usuário e uma composição da unidade de processamento da lógica e processamento do acesso ao banco de dados da aplicação. As interfaces para outras

aplicações tal como apresentado no item anterior devem ser bem definidas. Estas duas camadas lógicas de aplicações onde a interface de usuário é uma camada e lógica de programas e os serviços de acessos aos dados são uma única camada. Nesta categoria de aplicações os dados não podem ser acessados diretamente, eles podem ser acessados somente por funções pré-definidas. Muitas aplicações legadas existentes se enquadram nessa categoria. Nesses tipos de arquiteturas de aplicações os dados seqüenciais não podem ser trabalhados pelo usuário final e nem pelo administrador do data base, esta arquitetura de aplicações são também do tipo amigável [28].

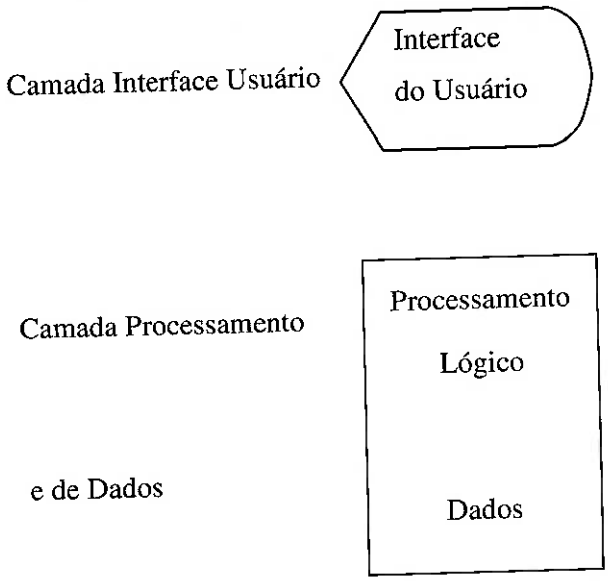


Figura 5 Categoria de Aplicações Legadas Decomposta por programa – Duas camadas

2.6.2.4 aplicações não estruturada ou monolíticas

Este tipo de aplicação legada tem as seguintes características: Todos os componentes da aplicação são componentes inseparáveis, e tanto a interface do usuário, da lógica do programa bem como os acessos aos dados são partes integrantes do mesmo componente programa. Essas aplicações de uma única camada, são geralmente aplicações muito velhas e sem estruturas. Em essência, os dados dessas aplicações podem somente ser acessados através das aplicações e através de terminais. Em geral, essas aplicações são hostis para a re-engenharia e são as mais difíceis em integrar e permitir uma reengenharia do legado.

Muitas aplicações podem ter uma arquitetura a qual poderá ter uma combinação desses quatro tipos de arquiteturas de sistemas legados. Isto permite identificar as arquiteturas existentes e assim ser feito um planejamento para gradualmente serem re-arquitetadas e migradas para serem integradas com novas aplicações ou reescrever se assim for mais atraente.

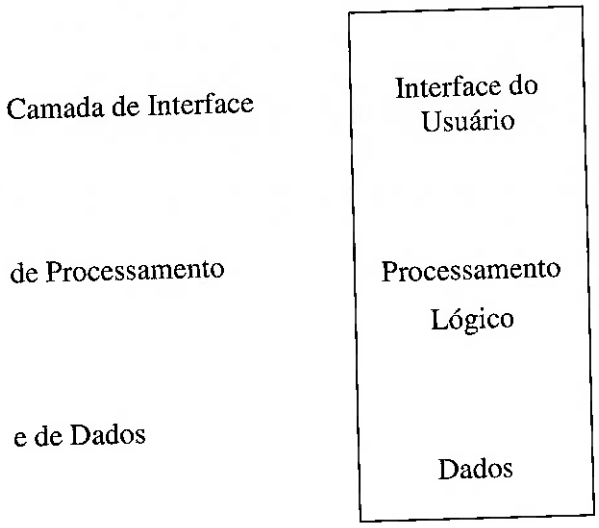


Figura 6 Categoria de Aplicações Legadas Monolítica - componentes em uma única camada

2.6.3 Fundamentos que propiciam a reengenharia de legados

A reengenharia de sistemas legados está centrada em princípios que norteiam os responsáveis pela área de TI para a tomada de decisão, o que fazer com os seus legados, se elabora um projeto para migrar os seus sistemas legados, se vai buscar no mercado soluções prontas para substitui esses sistemas legados ou se redesenvolver partir do **"zero"** novos sistemas para atender as necessidades da organização. Custo, tempo de execução do projeto, a importância dos sistemas legados para a organização são fundamentais para a escolha da alternativa mais apropriada. A seguir são apresentados esses princípios [28].

2.6.3.1 Valor de negócio da aplicação legada.

Para justificar qualquer reengenharia, as aplicações legadas deverão suportar os processos significantes atuais e futuros de um negócio. O valor de negócio pode ser medido em termos de contribuição para lucros em termos financeiros, e como um diferencial para os clientes, tipos de processos apoiados, e valor de mercado. Se necessário, os valores de mercado podem ser medidos numa pontuação entre 0 e 10 para indicar a importância dessa aplicação legada para a corporação.

2.6.3.2 Requisitos para desenvolvimento e flexibilidade

Se a aplicação não precisa ser modificada extensamente por questões de flexibilidade e crescimento, então o mínimo esforço para integrá-la com as novas aplicações pode ser apropriada. Requisitos para flexibilidade e crescimento podem ser medidos em termos de números melhorias funcionais e de performance necessárias para os próximos cinco anos (dois ou três anos tipicamente).

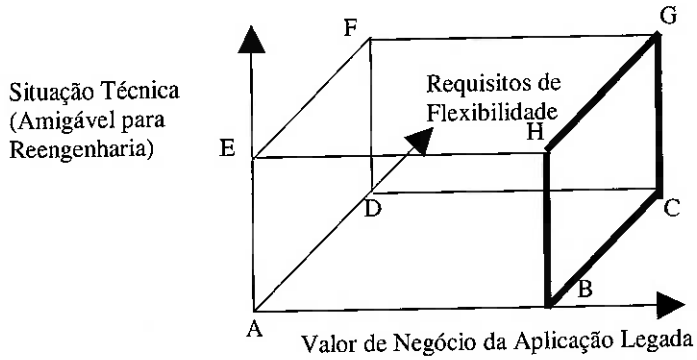
2.6.4 Qualidade técnica de uma aplicação legada

Se as aplicações legadas não são do tipo decompostas e se estão escritas em linguagem assembler utilizando com sistemas de arquivos convencionais e desatualizados do tipo indexados, a sugestão é melhor reescrevê-la nos pontos críticos. Situação técnica representa a qualidade da aplicação legada em termos de sua modularidade, taxas de erro, flexibilidade e utilização das tecnologias atuais. Qualidade técnica pode ser medida em termos de padrões internacionais de qualidade como ISO 9126. Informalmente, a situação técnica pode representar "hostilidade" da aplicação legada para reengenharia (aplicações monolíticas são mais difíceis de serem trabalhadas do que as que são as aplicações do tipo completamente decompostas).

A **Figura 7** mostra como esses três fatores contribuem para um primeiro entendimento de qual estratégia é a mais apropriada para reestruturar aplicações legadas. Em geral, as aplicações que agregam pouco valor ao negócio não deveriam ser consideradas para sistemas de reengenharia (essas são representadas pelo plano AEFD na Figura 7). Se necessária, a operação e manutenção dessas aplicações podem ser terceirizadas. Em um outro extremo,

uma ótima opção de tempo e empenho deve ser gasto nas aplicações que agregam altos valores ao negócio (o plano BCGH na Figura 7). As aplicações que têm baixo valor técnico (o plano ABCD na Figura 7) são os mais difíceis de lidar e custam muito caro para que a reengenharia seja feita. Essas aplicações são, geralmente, muito velhas e monolíticas. No outro extremo, as aplicações de alto valor técnico (o plano EFGH) são relativamente fáceis e não caras de fazer reengenharia e de se integrarem com aplicações novas. Aplicações legadas nessa categoria são tipicamente bem estruturadas aplicações baseadas em RDBMS. Finalmente, a dimensão da flexibilidade também se encaixa nessa análise. Aplicações legadas que não precisam se desenvolver dramaticamente (o plano **AEHB**) são boas candidatas para integração com novas aplicações enquanto as aplicações legadas com requisitos de alta flexibilidade e desenvolvimento (o plano CDFG) deveriam ser eventualmente reprojetadas e migradas.

O cubo exibido na **Figura 7** é uma ferramenta efetiva para a análise do "portifólio" da aplicação" para determinar a situação das aplicações atuais. Naturalmente, outros fatores podem ser, e deveriam ser, adicionados para análise futuro. A **Tabela 1** inclui fatores adicionais como pressões corporativas para reduzir os custos da aplicação, requisitos de consulta a dados, requisitos para circulação de dados, e requisitos de integração para sugerir caminhos na escolha da decisão apropriada (outros fatores podem ser adicionados a essa tabela). Essa tabela pode ser também usada como tabela de decisão. Ela mostra que a migração gradual pode ser escolhida se os requisitos de desenvolvimento e crescimento são altos e as pressões corporativas para reduzir os custos dos dados e da aplicação são altos também. Assim também, data warehousing pode ser apropriado se os requisitos de consulta a dados são muito altos e os requisitos de circulação de dados são baixos (ex. dados de planejamento); e o acesso no local pode ser apropriado para os dados os quais os requisitos de consulta a dados são baixos mas os requisitos de precisão dos dados são altos.



Nota: As linhas escuras deveriam ser a área principal de concentração. H = Ignore, B = Acesso no local, C = Migração (gradual ou cold turkey), G = Data warehousing

Figura 7 Esquema para Análise de Estratégias

2.6.5 Identificar estratégia para reengenharia de sistemas legados

Um fator importante no desenvolvimento de estratégias de migração de códigos é manter o sistema legado funcionando totalmente e sem restrições, enquanto durar o projeto de migração do sistema legado, não importando qual das estratégias tenha sido adotada. Alem disso para alcançar estes requisitos, certas metas e objetivos são inerentes para definir uma estratégia de componentização de sistemas, incluindo-se os seguintes aspectos [22]:

- **Minimizar os custos de desenvolvimento e organização**

Componentes migrados para conviver com códigos legados requer o desenvolvimento de adaptadores, pontes, e outros códigos de suporte que serão descartados depois de terminado a migração do sistema legado. Deve haver critério para o desenvolvimento desses códigos de suporte, pois representam uma despesa adicional, como este código deve ser projetado, desenvolvida, testada e mantida durante o projeto de migração. Minimizando o desenvolvimento de código de suporte é um modo de minimizar os custos gerais de desenvolvimento [22].

- **Suporte a um agressivo e previsível planejamento**

A estratégia de componentização deve procurar minimizar o tempo requerido para desenvolver e organizar os sistemas para serem reengenharia [22].

- **Manter a qualidade do produto**

Existem duas discussões sobre qualidade. Uma é a qualidade final. O sistema deve ser fácil de manter e implementar tecnologias que ainda não são obsoletas. A segunda é a qualidade em cada implementação de um componente [22].

- **Minimizar riscos**

Riscos podem ocorrer por diferentes maneiras, e são aceitos se ele é gerenciado e brando. Devido ao tamanho geral dos riscos e dos investimentos requeridos, é importante mante-lo baixo [22].

- **Manter a complexidade em nível gerenciável**

A estrutura caótica e tamanha de muitos sistemas legados é o maior fator de complexidade. Portanto, a estratégia deve minimizá-lo e manter o sistema em um nível adequado [22].

2.6.6 Estratégia de reengenharia de sistemas legadas

Existem muitas maneiras diferentes de se fazer reengenharia de sistemas legados.

Serão discutidas as mais importantes estratégias para um projeto de reengenharia de sistemas. A necessidade de um projeto de migrar sistemas legados é necessário que um projeto seja elaborado para definir as estratégias de reengenharia desses sistemas. As estratégias devem lidar com as aplicações legadas dentro das seguintes categorias [3], [17], e [11]:

Identificar uma estratégia para a reengenharia de software de sistemas legados de tecnologias orientadas a procedimentos (desatualizadas) para tecnologias orientadas a objeto, com o objetivo de recuperar, especificar, redesenhar, reescrever e reimplantar um sistema existente, do qual se conhece o código legado e eventualmente alguma documentação, é de fundamental importância para o processo de MIGRAR legados da tecnologia orientado a procedimentos para a tecnologia orientado a objetos.

No contexto de "software engineering" é muito importante diferenciar termos conceituais como as disciplinas "reengenharia" e "engenharia reversa", não apenas em quanto a

semântica, mas a sua essência na engenharia de software. E esse tema será abordado do ponto de vista conceitual a seguir.

2.6.6.1 Ignorar os sistemas legados

Descartá-las de todos os desenvolvimentos futuros. Por vezes os legados são trocados por soluções prontas do mercado e que levam anos e muito investimentos financeiros para serem "customizadas" para serem implementadas na organização.

Outra forma de ignorar legados é deixar os legados como estão, neste caso esta estratégia ignora complemente as necessidades da organização, não há muito que falar sobre essa alternativa, que por se só mostra o desinteresse da tecnologia da informação pelos negócios da organização, pois a tecnologia da informação é essencial para as organizações.

Essa estratégia é praticamente inviável na maioria das situações porque muitas aplicações legadas não podem ser ignoradas devido às suas bases de dados serem inter-relacionadas e de uso intenso por sistemas legados ou não. Em particular, os dados legados podem ser de mesmos valores, mesmo quando as regras de negócio (a funcionalidade) estejam desatualizadas. Por exemplo, muitas aplicações de telecomunicações novas necessitam ter acesso ao site e aos dados mais acessados do consumidor através das aplicações de telecomunicações legadas. Esse caminho é aconselhável quando as funções de negócios suportadas pela aplicação legada não serão utilizadas e estão sendo retiradas, o status técnico da aplicação é alto o suficiente para minimizar os custos com manutenção, os requisitos para desenvolvimento e flexibilidade são nulos, e o que precisa para integração com outras aplicações é muito pouco.

2.6.6.2 Reescrever

Esta estratégia poderá incorrer num problema que é muito comum nas organizações. Nem sempre existe a documentação do sistema legado, ou se existe não é confiável e sendo assim não pode dar uma base de sustentação para o redesenvolvimento de um sistema. Um outro fator de risco para essa alternativa são as muitas regras do negócio embutidas no código e a falta do conhecimento dessas regras pelo pessoal da tecnologia da informação. Pessoas que

participaram ou desenvolveram esses sistemas estão em outras funções ou não estão mais na organização.

Essa tática também não é viável em muitas situações práticas porque não é fácil reescrever sistemas de aplicações legadas a partir do **"nada"** devido aos seguintes fatores [28].

- Um sistema melhor deve ser prometido ao usuário porque muito provavelmente o custo benefício não justifique grandes investimentos somente pela promessa de flexibilidade.
- As pressões sobre os negócios não esperam para permitir novos desenvolvimentos (para algumas aplicações legadas, de 5 a 10 requisições de mudança são recebidas por mês).
- As especificações para sistemas legados raramente existem além de muita dependência sobre o sistema legado e que não estão documentadas.
- Os esforços para reescrever uma aplicação legada podem ser muito grandes e exigem forte gerenciamento.
- Reescrever grandes aplicações legadas exige muito tempo (muitas das novas tecnologias se tornam legadas de três a cinco anos).

Como descrito, essa tática deve ser considerada por aquelas aplicações com grande expectativa de vida, e com alta demanda para flexibilidade, redução de custos, e integração com outras aplicações. Terceirizar a tarefa para reescrever uma aplicação legada é uma opção viável para muitas organizações. Em particular, a terceirização para fábricas de softwares pode ser vantajosa do ponto de vista custos. Dedene e DeVreese [10] relatam dois casos de estudo que exibem as vantagens da reengenharia por terceiros.

2.6.6.3 Integrar

Consolidá-las nas aplicações atuais e futuras adequadamente. Essa alternativa é um misto do novo com o velho. Significa integrar as novas tecnologias com as tecnologias velhas, o que é possível. Decidir por essa alternativa impõe a organização a se "perpetuar", nunca ou quase isso, a sair de tecnologias velhas, pois a interação de sistemas novos com base de dados de tecnologias velhas são constantes e normais. A não ser que essas bases de dados sejam duplicadas para serem usadas pelos sistemas em novas tecnologias. Aí pode incorrer em

duplicidade de informação, de inconsistências de dados. É necessário uma profunda avaliação para aplicar essa alternativa.

Esta estratégia deve ser usada nas seguintes situações. (ver tabela 1): O acesso aos dados necessários é urgente e não podem ser adiados até o término da migração.

- Valor agregado dos dados e dos códigos da aplicação são altos.
- Uso dos dados enquanto uma migração completa deve ser feita.
- Necessidade de acesso a dados por aplicações cliente e ferramentas de usuário final
- Queries envolvendo poucas consultas aos dados
- Exige alto volume de movimentação de dados são altos, usuários necessitam acessar a cópia mais recente dos dados, não um data warehouse ou banco de dados desatualizados.

2.6.6.4 Data warehouse

Construir um sistema como "proteção" para hospedar os dados freqüentemente acessados. Não deve ser considerado como uma estratégia para a reengenharia de sistemas legados e nem para implementar novas tecnologias nas organizações, pois o objetivo maior numa reengenharia de sistemas é a implementação de melhorias no código, (já discutidas anteriormente), e transformá-lo num aplicativo orientado a objetos, e essa alternativa chega apenas aos dados mais freqüentemente utilizados. Duplicar a base de dados mais utilizados e desenvolver qualquer aplicativo sobre essa base, é oficializar a inconsistências de dados na organização.

Essa alternativa deve ser usada quando a organização entender o data warehouse como um modelo poderoso de banco de dados que aumenta significamente a capacidade dos administradores de analisar rapidamente conjunto de dados extensos e multidimensionais.

O conceito de Data warehouse é baseado na notação que dados organizacionais existem em dois formatos:

- Dados operacionais os quais são usados para o processamento das transações do dia-a-dia. Esses dados são atualizados freqüentemente.

- Dados de análise (suporte de decisão) os quais são usados para análise de negócio e geração de relatórios. Esses dados são extraídos dos dados operacionais periodicamente e recebidos, geralmente, por outro computador para geração de relatório e análise.

2.6.6.5 Migração Gradual

Reprojetar, redesenvolver e fazer a transição gradualmente. Essa é a alternativa mais adequada quando se trata de migrar sistemas legados numa organização. Esta alternativa permite de fato a implementação de novas tecnologias numa organização, em todos os aspectos, podendo envolver tecnologias de bancos de dados, implementar sistemas orientados a objeto e abrir porta para sistemas WEB de forma nativo. É sem dúvida a alternativa mais atraente, pois além das tecnologias que possam ser implementadas, poder-se melhorar os sistemas existentes.

As estratégias de migração graduais de aplicações legadas são geralmente mais eficientes que reescrever a partir do início (cold turkey). A estratégia de migração gradual envolve muitos passos durante o período de muitos anos e usa "tecnologia de ponta" para fazer a transição da aplicação legada sem que o cliente e o usuário final percebam.

Os sistemas legado e migrado "target" precisam coexistir durante o estágio de migração. O desafio principal é criar uma abertura para isolar as etapas de migração, para poupar o usuário final e para que ele não perceba se a informação requisitada está sendo recuperada de um módulo/banco de dados antigo ou de um módulo/banco de dados novo. Durante a migração, deve prover uma abertura para [4]:

- Acesso a funções da aplicação legada ou dados de interfaces do usuário alvo;
- Acesso a funções da aplicação alvo ou dados das interfaces legadas do usuário;
- Esconder através de encapsulamento, detalhes do legado e dos sistemas alvo, dos usuários [8];
- Sincronização dos dados alvo (target) e legados;

Desenvolvimento de software (gateway) para facilitar o processo de migração é por vezes caro. O custo total, o tempo do projeto, o pessoal envolvido e as considerações

organizacionais necessárias para a migração precisam ser consideradas cuidadosamente, pois pode inviabiliza o projeto de migração[28].

Fatores de Avaliação	Ignorar o Sistema Legado	Reescrever do zero	Integrar (Acesso no Local)	Data Warehousing	Migração Gradual
Valor de negócio de uma aplicação legada	Baixa (curta extensão/não crítica)	*Alta (extensão muito longa)*	Alta	Alta	Alta
Requisitos para flexibilidade e desenvolvimento	*Nula para muito baixa*	Muito alta e imediata	Baixa	Média	Alta
Situação técnica da aplicação legada	Alta (decomponível)	Muito baixa	Baixa para média	Média	Média para baixa
Pressão corporativa para reduzir custos e sistemas cliente/servidor	Baixa	Alta e imediata	Baixa	Média	Alta e longa extensão
Requisitos para consulta a dados (direto ao assunto)	Baixa	Média	Baixa	Alta para muito alta	Média
Requisitos para circulação de dados			*Alta*	Baixa	Média
Integração necessária com outras aplicações servidora	Baixa para muito baixa	Alta para muito alta	Baixa	Baixa	*Alta para muito alta*
Número de aplicações locais acessadas por clientes para os dados necessários			Pouca	Larga	

TABELA 1 DIREÇÕES GERAIS PARA LIDAR COM UM SISTEMA LEGADO [28]

CAPÍTULO 3 MODELO PARA REENGENHRAIA DE SISTEMAS LEGADOS

Identificar um modelo de reengenharia de software para migrar sistemas legados de tecnologias orientadas a procedimentos "OBSOLETAS" para tecnologias orientadas a objeto com o objetivo de recuperar, especificar, redesenhar e reimplantar um sistema existente, do qual se conhece o código legado e eventualmente alguma documentação, é fundamental para o processo de reengenharia de software.

Também, uma motivação para a aplicação da reengenharia de sistemas legados é a diminuição dos altos custos de manutenção de sistemas que se devem a diversos fatores [29]. A manutenção contínua faz com que a implementação torne-se inconsistente com o projeto original, o código torne-se difícil de se entender e sujeito a muitos erros. Além da documentação desatualizada. As linguagens de programação em que esses sistemas legados foram implementados estão ultrapassadas, não havendo suporte por parte dos seus fabricantes e o ensino destas linguagens pelas universidades não tem a mesma importância que as linguagens orientadas a objetos possuem, contribuindo dessa forma, para a escassez de profissionais que as dominem.

Assim, a reengenharia torna-se uma área interessante para ser explorada dentro do campo da engenharia de software, que objetiva melhorar a produtividade e qualidade dos processos de "modernização" de software. Para cumprir tal objetivo a reengenharia de software precisa entender corretamente os programas existentes, a partir dos códigos fontes e documentação disponíveis, para facilitar a realização de mudanças e reconstrução do software numa nova tecnologia. Portanto, o propósito da reengenharia de software é facilitar a execução de mudanças e correções, a recuperação do desenho em um nível mais alto de abstração, o redesenho e a reprogramação de um software.

São apresentados as metodologias, os processos e etapas que possibilitam as organizações a elaborarem um projeto para migrar os seus sistemas legados orientados a procedimentos para tecnologias de sistemas orientados a objeto, de forma segura, pois uma estratégia mal conduzida, pode causar "estragos" na área de tecnologia da informação não apenas do ponto de vista custo financeiro de migração, mas também desacreditar junto a área de TI estas

tecnologias, bem como a capacidade técnica das pessoas envolvidas em um projeto dessa natureza.

São apresentadas também alternativas para as estratégias para a migração desses sistemas legados nas corporações através de alternativas discutidas nesse capítulo. Essas alternativas pressupõem a utilização da reengenharia, como forma de atualizar sistemas legados, sem ter que redesenvolver esse sistemas ou substituí-los por "pacotes".

Como resultado desse trabalho serão apresentadas modelos para um projeto de reengenharia de sistemas legados, tais como: requisitos, planejamento, arquitetura, reorganização do código, reescrever o código, testes, validação e implantação dos sistema legado em produção.

3.1 ETAPAS PARA A REENGENHARIA DE SISTEMAS LEGADOS

O sucesso do projeto está condicionado a uma série de atividades - as etapas para a reengenharia de sistemas legados - as quais devem ser identificadas e elaboradas de forma organizada em projeto para migrar os sistemas legados de uma na corporação. Essas atividades são fundamentais e ocorrem em todas as fases do projeto. Nos capítulos a seguir são elencados essas atividades que não podem ser desconsideradas, quando do aceite do projeto junto aos executivos da corporação e que foi à base dos estudos para a reengenharia de legados, e que serão tratados nos tema a seguir [28]:

A figura 8 apresentada a seguir, resume o processo de reengenharia de sistemas legados, onde as etapas são identificadas e discutidas: A etapa Análise de requisitos permite que no processo de reengenharia sejam efetuadas entre outras, mudanças na estrutura de dados, ou na API, etc. Na etapa Planejamentos são discutidos os sistemas candidatos, prioridades, tempo e recursos a serem utilizados. Na etapa Arquitetura são definidas as tecnologias a serem empregada bem como a arquitetura para a qual os sistemas serão feitos a reengenharia. Na etapa Reorganização do código é a etapa em que serão tratadas as reorganizações do código para permitir a sua reengenharia. A fase Conversão do código é a etapa em que o código será convertido para as tecnologias adotadas e devem ser feitas de modo automatizado. N etapa de testes, os sistemas migrados dever ser testados exaustivamente para a sua homologação e implantação. Na etapa de implantação, deve ser discutido treinamento, e implantação da aplicação migrada para a tecnologia adotada.

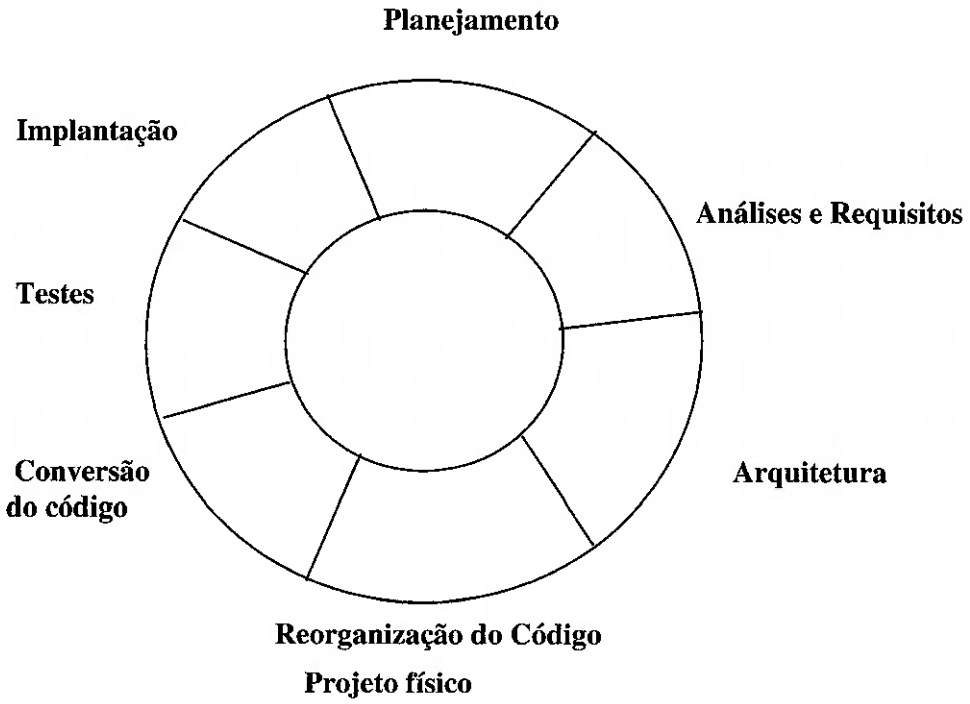


Figura 8 Modelo do processo de reengenharia de sistemas

3.2 PLANEJAMENTO PARA REENGENHARIA DE SISTEMAS LEGADOS

Uma etapa importante no processo de reengenharia de sistemas legados é o planejamento das atividades a serem executadas, sendo que um dos objetivos é fornecer um esboço de forma a permitir estimativas dos recursos a serem utilizados, bem como dos custos e de um cronograma para as atividades de reengenharia. Essa estimativas deverão ser feitas dentro de uma perspectiva determinada de tempo e devem naturalmente ser atualizadas na medida em o projeto avance. Também deverá ser estimado o cenário para delimitar os melhores e piores casos para o comportamento do projeto. Outro objetivo do planejamento é sincronizar a reengenharia de um sistema legado, com os objetivos de negócios da corporação.

A seguir são apresentadas as atividades associadas ao planejamento de um projeto para a reengenharia de sistemas legados e que envolvem: [28].

➢ **Sistema legado como motivação do negócio e a expectativa da reengenharia**

Reengenharia de sistemas legado é um processo caro e arriscado. A finalidade dessa

atividade, é um esforço para entender e analisar as bases e motivação dos negócios e em que os sistemas legados agregam valores aos negócios da corporação, antes de começar o processo de reengenharia. Essa análise deve considerar os seguintes aspectos:

- Deve-se estabelecer as expectativas da reengenharia no negócio da corporação;
- Analisar as oportunidades de negócios;
- Alinhar a Tecnologia da Informação com os negócios da corporação .

➢ **Avaliar a arquitetura para o sistema legado**

Avaliar uma arquitetura para atender as necessidades da corporação, especialmente para atender as estratégias dos negócios desta, elencando-se as aplicações candidatas para o processo de reengenharia, considerando os seguintes aspectos:

- Buscar uma arquitetura mais apropriada, que atenda as necessidades da aplicação e
- leve em consideração custos, manutenibilidade e disponibilidade;
- Escolher uma estratégia mais apropriada para a reengenharia da aplicação;

➢ **Avaliar e planejar a infra-estrutura da arquitetura**

Avaliar e planejar a mais apropriada infra-estrutura da arquitetura (plataforma) para determinar as tecnologias necessárias para desenvolver o projeto de reengenharia de sistemas legados e deve-se considerar as seguintes atividades:.

- Avaliar as necessidades de infra-estrutura de Tecnologia da Informação
- Escolher a mais apropriada infra-estrutura de Tecnologia da Informação.

➢ **Analisar o custo benefício**

A principal finalidade da análise do custo benefício é **comparar** os benefícios esperados com: a estimativa de tempo, os recursos financeiros despendidos, o hardware e software envolvidos e recursos humanos necessários para a execução de um projeto de reengenharia de software legado. Os seguintes tópicos são essenciais nessa atividade:

- Estimar o esforço necessário para o desenvolvimento e suporte do projeto;
- Um esboço dos custos envolvidos previstos e imprevistos e os benefícios que se busca;
- Calcular se os custos são atraentes, ao menos, equivalem aos benefícios esperados.

➢ **Determinar custos e migração de aplicação.**

Determinar se o custo / benefício da migração é favorável para uma aplicação ser candidata a reengenharia. Isto envolve verificar no mercado se há soluções prontas que atenda a troca do legado por "pacotes" e os custo sejam atraentes, ou se o tempo e o custo para redesenvolver justificam o investimento para o desenvolvimento de um novo sistema, em substituição ao legado. Pode ser avaliado também, e nessa tese é que são encontrados os grandes benefícios da reengenharia: Apesar do código ser "velho" a tecnologia possa estar ultrapassada, mas a regra do negócio continua firme e atendendo plenamente aos requisitos funcionais da aplicação e do negócio.

➢ **Avaliar impacto da arquitetura na organização envolvendo – HW e SW**

Identificar o ambiente de execução na qual a aplicação migrada será migrada. Descrever o impacto, e diferenças entre este ambiente "alvo" e o ambiente de execução da aplicação original, relativos a arquitetura, Hardware e software.

➢ **Criar um ambiente de desenvolvimento de reengenharia.**

Criar o ambiente de desenvolvimento e disponibilizar ferramentas necessárias para fazer a reengenharia da aplicação legada.Este ambiente deve contemplar o hardware e software envolvido bem como um ambiente de simulação, para permitir a comparação dos resultados obtidos dos sistemas legados com os resultados obtidos na nova tecnologia.

- **Identificar necessidades de um projeto de reengenharia piloto**

Devido ao grande número de fatores no ambiente legado e o número de usuários que são potencialmente afetados, implementações piloto em pequena escala podem ser recomendadas para dar ajuda nos esforços de migração. Esta atividade deve conter a extensão para a qual as considerações de migração sugerem que as soluções piloto são necessárias para validar a integridade do sistema, performance e aceitação dos usuários. Embora tenha os seus valores em um teste de solução técnica, pilotos também servem como um mecanismo para prover informes das entradas dos usuários e para alcançar a confiança do usuário. Depois de determinado o escopo do piloto é necessário estabelecer o grau de envolvimento do usuário, validações em verificações.

- **Treinamento na tecnologia alvo**

Um treinamento adequado é um passo muito importante para melhorar a produtividade e a qualidade do trabalho de uma equipe. Em Informática, isso vale tanto para o domínio de técnicas já consagradas como para a atualização constante em novas tecnologias e produtos.

Um projeto que envolva migrar sistemas legados, não contempla apenas o código legado, as regras de negócio, mas também as tecnologias obsoletas empregadas nesse projeto. Migrar esse legado envolve também profissionais da Tecnologia da Informação e envolve usuários. Elaborar um projeto dessa natureza exige um adequado treinamento para as tecnologias alvo "target" para os profissionais envolvidos, a fim de que o projeto possa fluir com naturalidade e sem tropeços e possa ser implementado com sucesso na organização, pois um projeto dessa magnitude envolve custos vultosos e sinergia de toda uma equipe de projeto e perspectiva de solução dos problemas existentes. A falta de um treinamento poderá frustrar e os prejuízos são muito grandes.

É provável que muitos profissionais experientes na área de desenvolvimento de sistemas não se adaptem ao estilo exigido pelo paradigma de orientação a objetos, como alvo de tecnologia "target" num processo de migrar sistemas legados.

Como as empresas tendem a usar os seus melhores profissionais nesse tipo de desafio tecnológico, corre-se o risco de que, por falta de condições de adaptação, esses profissionais se utilizem "velhos hábitos" de desenvolvimento de sistemas, subtilizando os benefícios da orientação a objetos e levando as organizações a conclusões errôneas sobre a validade desta tecnologia. Para minimizar esse impacto, é imprescindível o uso de uma metodologia e um eficiente processo de reeducação e treinamento [18].

Tomando a reeducação e treinamento como fatores básicos para iniciar qualquer processo de mudança dentro de uma organização, uma primeira questão geral que se deve analisar está associada aos fatores que facilitariam ou dificultariam o processo de aprendizagem de um novo modelo de desenvolvimento e ou manutenção de sistemas. As mudanças aceleradas na área de informática fazem com que as organizações, as empresas de desenvolvimento de sistemas mantenham em suas equipes profissionais com formação e experiência cada vez mais heterogêneos, configurando um conjunto de profissionais com perfis diferenciados com reflexo direto no processo de aprendizagem de novos modelos e técnicas.

O impacto que a tecnologia orientação a objetos causa nos profissionais mais experientes na área de tecnologia da informação, está relacionado à resistência desses profissionais, formadas ao longo dos anos no paradigma da tecnologia de desenvolvimento e manutenção de software estruturado e orientados a procedimentos, em abandoná-lo para adotar o novo paradigma, uma vez que, este ainda tem se mostrado satisfatório para solucionar a maior parte dos seus problemas de manutenção e desenvolvimento de sistemas [31]. Esta resistência de certa forma pode estar associada a dificuldade em assimilar e aplicar de forma correta os conceitos de OO. Para isso, é necessário que as pessoas sejam capazes de pensar em termos de objetos, suas ações e as colaborações entre esses objetos. Experiências percebidas nas organizações demonstram que usualmente as pessoas utilizam metodologias e linguagens de programação orientados a objetos, mas continuam trabalhando da mesma forma estruturada, definindo e mantendo softwares em termos de procedimentos, funções e estruturas de dados [1] , [2] e [31].

- **Controle sobre o legado**

Em toda organização espera-se que a área de TI tenha o mínimo de controle sobre o seu legado, envolvendo aplicativos e dados. Sabe-se que desde as grandes organizações até as pequenas, esse controle pode até ser rigoroso, mas como a idade dos sistemas legado variam

de sistemas recém implantados a até mais de 20 anos de existência, esse controle pode não representar a verdadeira situação desses legados.

Sabe-se que há um misto de falta de códigos fontes e de até várias versões de código fonte de um mesmo programa. A precisão de informação sobre os sistemas legados que permitam delinear o perfil mais realista da empresa é necessário para elaboração do projeto, pois os custos, tempo para migração são determinados a partir dessa informações. Se a organização não dispuser dessas informações, será necessário provê-las antes do início do projeto para que seja possível efetuar o planejamento e quantificar os recursos humanos, financeiros e usuário envolvidos.

- **Envolver a alta administração da empresa**

A reengenharia de sistemas legados deve ser encarada como um projeto da organização, não como um projeto pessoal e tão pouco só da TI. Assim sendo, tendo-se essa consciência os responsáveis pela Tecnologia da Informação deverão envolver toda a organização, assumindo junto à alta administração, bem como dos seus pares todo o ônus do projeto. Os recursos financeiros, os recursos humanos, os prazos para a execução do projeto, a priorização das áreas que deverão ter os seus sistemas legados migrados deverão ser objeto de negociação e compromissos firmes. Os recursos financeiros deverão fluir com naturalidade, para que não haja interrupções no decorrer do projeto e o sucesso seja o objetivo final.

- **Construir um processo de confiança junto ao usuário**

Os usuários deverão ser envolvidos de forma amigável, para que seja possível estabelecer um regime de confiança e ter o usuário como um aliado no projeto, pois sem o apoio deste, o projeto tende a demorar muito e ter os seus prazos e custos comprometidos. A partir desse compromisso de confiança estabelecido, muitos dos problemas que estavam **"hibernando"** , como funções do sistema legado que não estavam funcionando, podem ser corrigidos durante o processo de reengenharia de sistemas e requisitos dos usuários, que caíram no esquecimento podem ser recuperados e atendidos.

3.3 ANÁLISES E REQUISITOS DE REENGENHARIA DE SOFTWARE

A escolha de um padrão para extrair o conhecimento dos códigos fontes legados,

principalmente os de terceira geração, como, por exemplo, o COBOL, constitui-se de um método de engenharia reversa que, visa facilitar o processo de reengenharia de sistemas, para a recuperação de informações úteis ao entendimento do software. Através desse método é possível a recuperação de visões funcionais e de visões estruturais do sistema. As considerações lógicas são obtidas por meio da análise da interface para a recuperação de visões funcionais do sistema e, posteriormente, parte-se dessas visões recuperadas e de considerações físicas obtidas por meio da análise do código fonte, para a recuperação de visões estruturais do sistema. A maior dificuldade a ser encontrada nesse processo, nas linguagens do tipo procedurais, é o uso indiscriminado de comandos de "desvio", como por exemplo o comando "GO TO....", comandos de encapsulamentos de desvios "ALTRER" que altera dinamicamente uma lógica de programa, REDEFINIÇÕES de estruturas de dados, na linguagem Cobol. Todos esses desvios devem ser tratados e retirados do código, pois programas orientados a objeto, em JAVA não suportam esses procedimentos.

A seguir são apresentados os principais requisitos que devem estar aderentes ao processo de reengenharia de sistemas legados.

3.3.1 Conservar a familiaridade entre os sistemas e usuários

A familiaridade operacional da navegabilidade e, da usabilidade devem ser sempre preservadas e, quando isso não for possível, deverá ser documentada para se for o caso retreinar o usuário nas funções de usos do sistema novo, quando da sua homologação. Quando isso acontecer, as mudanças deverão ser formalmente negociadas entre as pessoas da área de Tecnologia da Informação e os representantes do usuário final, que deverá dar o seu aceite.

3.3.2 Remodularizar o código legado com funcionalidades em duplicata.

Remodularizar o código da aplicação legada para separar e retirar as funcionalidades duplicadas dos códigos das funcionalidades da aplicação.

3.3.3 Requerimentos funcionais

É muito importante também que no processo de reengenharia de sistemas legados, os seguintes requerimentos funcionais sejam fixados [28]:

- Tempo de resposta;
- Segurança dos dados;
- Restringir a disponibilização de dados;
- Possibilitar a conectividade com outros ambientes (sites);
- Interoperabilidade e interfaces com outros sistemas;
- Portabilidade de plataformas de Hardware/ Software (processamento, apresentação e dados);
- Total controle de manuseio de cópias de segurança e recuperação de dados;
- Manutenibilidade do sistema;
- Uso ilimitado de acessos aos Bancos de dados por qualquer número de usuários autorizados;
- Políticas de restrições (Padrões e políticas a ser seguido, controles de aplicações, auditar restrições)

3.4 ARQUITETURA – AMBIENTE PARA REENGENHARIA DE SISTEMAS LEGADOS

Criar um ambiente operacional para a reengenharia de sistemas legados com as características da arquitetura em que o sistema irá operar, é a atividade que antecede a reengenharia

propriamente dita. Nesta etapa, todo o ambiente deverá estar disponibilizado para as etapas seguintes, que são a reengenharia, os testes, a validação do projeto e a implantação do sistema legado na nova tecnologia.

Outras atividades de preparação para o ambiente, deverão ser objeto de trabalhos reparatórios para a reengenharia dos sistemas, dentre elas uma revisão de atividades executadas desde o início do projeto.

São apresentadas a seguir as etapas para a montagem do ambiente operacional onde a reengenharia de sistemas será executada, para migrar sistemas legados para tecnologias orientados a objeto.

3.4.1 Montagem do ambiente operacional

Esta atividade tem por objetivo a criação da infra-estrutura necessária para as atividades de reengenharia de sistemas, e deve contemplar:

- Instalação de Hardware ;
- Instalações de Software;
- Customização de Hardware e de Software definidos no início do projeto;
- Criar infra-estrutura operacional para a equipe de reengenharia;
- Integrar esse novo ambiente ao ambiente existente;
- Caso seja necessário treinar a equipe de reengenharia.

3.4.2 Análise de inventário

Deverá ser elaborado um inventário dos sistemas legados, contemplando: uma descrição detalhada, contendo o tamanho, idade dos sistemas, e a criticalidade para cada aplicativo que for para a reengenharia. Essas informações devem ser ordenadas de acordo com a criticalidade para o negócio da organização, a longevidade e manutenibilidade do sistema para a organização.

3.4.3 Reestruturação da documentação dos sistemas

A grande maioria dos sistemas legados têm pouca documentação. Na medida do possível procurar recuperar a documentação quando não houver e procurar documentar todos os sistemas, se houver recursos e tempo para essa finalidade.

3.4.4 Revisão do projeto de reengenharia de sistemas

Uma revisão do projeto de reengenharia de sistemas proporciona uma comparação do projeto em andamento com o que foi projetado, e se houver desvios, deverão então ser corrigidos. As revisões dos principais itens são apresentados a seguir [14] , [15]:

- Metas e objetivos da organização com relação aos sistemas legados;
- Prioridades dos sistemas e necessidades dos usuários;
- Recursos financeiros disponíveis;
- Requisitos não funcionais (desempenho, segurança, interoperabilidade);
- regulamentos, políticas e padrões relevantes, e regras/doutrinas de negócio.

3.4.5 Administrar esforço de migração – migrar, manter e controlar retrabalhos

Administrar o esforço de migração é um fator crítico de sucesso. É importante determinar como melhorias em processos nos sistemas legados serão administradas enquanto o sistema alvo está em processo de reengenharia.

A abordagem da administração da migração deve considerar como:

- Controlar as manutenções, enquanto o sistema está em processo de reengenharia;
- Controlar retrabalhos de reengenharia;
- Acompanhar progresso e pontos de controle;

- Identificar e monitorar questões em aberto;
- Identificar e reduzir riscos;
- Estabelecer comunicação entre desenvolvedores de sistemas, pontos de contato para sistemas legados, organizações responsáveis por sistemas com interface, clientes e grupos de usuários.

Os sistemas legados e migrado "alvo" precisam coexistir durante o estágio de migração. O desafio principal é criar uma abertura para isolar as etapas de migração, para poupar o usuário final de dupla atividade e confusões operacionais, e para que ele não perceba se a informação requisitada está sendo recuperada de um módulo/banco de dados antigo ou de um módulo/banco de dados novo. Durante a migração, deve-se prover uma abertura para [4]:

- Acesso a funções da aplicação legada ou a dados de interfaces do usuário alvo;
- Acesso a funções da aplicação alvo ou a dados das interfaces legadas do usuário;
- Esconder através de encapsulamento, detalhes do legado e dos sistemas alvo, dos usuários [8];
- Sincronização dos dados alvo (target) e legados;

O desenvolvimento de softwares tradutores (gateway) para facilitar o processo de migração é por vezes caro. O custo total, o tempo do projeto, o pessoal envolvido e as considerações organizacional necessárias para a migração, precisam ser consideradas cuidadosamente, pois podem inviabiliza o projeto de migração[28].

3.4.6 Criar protótipos

Protótipos podem, efetivamente, testar soluções em potencial, especialmente em casos onde os sistemas corrente são complexos e envolvem muitos usuários. O plano de migração deve identificar a necessidade de prototipação. Ao mesmo tempo, deve endereçar a extensão da necessidade da migração para reduzir o risco de migração e de demonstrar a adequação de conceitos.

Esta atividade deve também endereçar o escopo da necessidade de prototipação, os conceitos de migração que estão sendo testados, o conjunto de resultados esperados, e mecanismos para avaliar se os resultados esperados foram obtidos.

Os protótipos devem ser significativos, e precisam ser mais que programas de demonstração para relações públicas ou de marketing. Soluções de protótipos podem ser avaliados através de uma variedade de significados, incluindo avaliação de prova de conceitos, avaliação de usuários, e avaliação de arquitetura.

A interface de usuário é particularmente apropriada para prototipação. Por exemplo, um protótipo "story-board" provê uma tela simples que imita o que o usuário deveria perceber no sistema. O protótipo pode ser finalizado em semanas em vez de meses de especificação trabalhosa. Um protótipo efetivo pode também permitir que usuários façam uso do cenário operacional da nova interface, antes das decisões de implementação de software e hardware serem consolidadas.

3.5 REORGANIZAR O CÓDIGO LEGADO – PROJETO FÍSICO

Na prática, a tentativa de conversão direta de programas codificados em uma linguagem para outra (*translation*), resulta na preservação de todos os problemas da linguagem original e adiciona novas complicações causadas pela linguagem diferente. Pode-se dizer que superar esta diferença é a parte mais difícil da reengenharia.

A extração automática de objetos de um programa Cobol corresponde a responder a questão de como o programa poderia ser escrito usando linguagens mais modernas e ferramentas adequadas. A transição para a tecnologia OO é dividida por ferramentas em duas fases. A primeira fase trata da reestruturação do programa fonte, típico em ferramentas de reengenharia. Após a reestruturação, o programa estruturado se torna transparente e mais fácil de manter (até 44 % C. Babcock). A próxima etapa é a transformação do programa reestruturado em um programa orientado a objeto. A abordagem óbvia é a divisão do programa em classes, difícil de formalizar e automatizar.

3.5.1 Reestruturação do código - primeira fase

Naturalmente, diferenças entre as linguagens original e final complicam a reestruturação, sendo proposto a uma reestruturação em fases, a ser aplicada em programas escritos na linguagem COBOL.

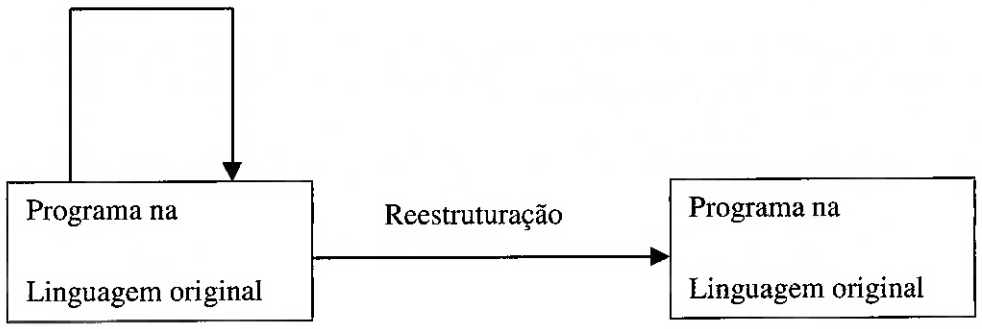


Figura 9 Reestruturação de código

Em caso específico para linguagem COBOL, a reestruturação a ser feita por ferramentas envolve os seguintes componentes:

- **Preparar o código Cobol para a criação de Métodos e Sub Métodos**

A linguagem COBOL não dispõe de "Método" ou "Sub-método". O único meio de estruturação de um programa na linguagem Cobol é via "Parágrafos" ou "Section' dentro da Procedure Division. É necessário, conseqüentemente transformar os parágrafos de programas escritos em Cobol em Sub-Métodos e as Sections do programa Cobol em Métodos para possibilitar o processo de conversão para linguagens do tipo Orientadas a Objetos".

- **Tratamento de Operadores**

Localização ou eliminação de GOTO e ALTER. Na época da criação do Cobol os desvios incondicionais eram perfeitamente aceitáveis, muito embora esses comandos dificultassem o entendimento de um programa Cobol. Algumas linguagens, como **JAVA** por exemplo, eliminaram completamente esses operadores GOTO e ALTER. Na reengenharia deve-se

converter GOTOs para outros operadores estruturados, com semântica equivalente, e buscar uma solução para resolver o problema do ALTER.

- **Tratamento de redefinição de estruturas**

Localização ou eliminação de REDEFINES. Esta facilidade para a linguagem COBOL é muito utilizada em estruturas de arquivos convencionais, não sendo suportados em tecnologias OO e nem tampouco em estruturas de Bancos de Dados Relacionais. Deve-se normalizar as estruturas de arquivos que utilizam essas facilidades.

- **Tratamento de Arrays**

Localização e tratamento de ARRAYS. Esta facilidade para a linguagem COBOL é muito utilizada em estruturas de arquivos convencionais e em áreas de trabalho e devem ser organizadas para ser suportada na nova estrutura de linguagens em tecnologias OO. Deve-se normalizar as estruturas destes ARRAYS para serem melhores utilizadas.

- **SORT interno**

A alternativa para tratar o SORT interno, para os casos de pouco volumes de dados a serem classificados deverá ter o seu tratamento feito em memória. Para grandes volumes deverá ser externalizado em arquivos externos em disco.

- **Chamadas a sub-rotinas externas**

Deverão ser transformadas em objetos a serem utilizados pelos programas que passarem pelo processo de reengenharia de software.

- **Localização de Dados**

Uma das grandes diferenças entre Cobol e linguagens como o JAVA é que todos os campos de dados são globais. Para fazer com que o programa transformado esteja conforme com a ideologia da linguagem destino, há necessidade de localizar os dados. Uma passagem é usada

para distribuir, otimizadamente, as variáveis entre as procedures e organizar a transmissão de variáveis para os locais necessários como parâmetros.

3.5.2 Otimização das transformações

Sistemas legados podem ter sofrido manutenções durante décadas, que transformaram de tal forma o projeto original que a estrutura atual nem de longe se parece com a original. As otimizações das transformações focalizam aquelas que facilitam as manutenções futuras, principalmente, variáveis e códigos não utilizados.

3.5.3 Código legado Reorganizado

A seguir são apresentados códigos Cobol, reorganizado de tal forma que possa ser utilizado como entrada para ferramentas que reescreva o novo código na linguagem destino, a partir do código organizado.

A seguir é apresentado de forma Procedimental um modelo que, a partir de um programa COBOL, do tipo monolítico (uma camada) possa ser migrado para o modelo em três camadas.

Modelo para a extração de código legado Cobol

```
                                  +---------------------------+
                                  | IDENTIFICAÇÃO DE ARQUIVOS |
                                  +---------------------------+
|ID  DIVISION.                                  |
|PROGRAM-ID. FBNH2010.                          |
|AUTHOR. JAPI.                                  |
|*REMARKS.                                      |
|ENVIRONMENT DIVISION.                          |
|CONFIGURATION SECTION.                         |
|SPECIAL-NAMES.                                 |
|   DECIMAL-POINT IS COMMA                      |
|   C01 IS CANAL-1.                             |
|INPUT-OUTPUT SECTION.                          |
|FILE-CONTROL.                                  |
|SELECT ENTDATAS ASSIGN UT-S-ENTDATAS.          |
| SELECT ENTCADAS ASSIGN UT-S-ENTCADAS.         |
| SELECT SAILISTA ASSIGN UT-S-SAILISTA.         |
|DATA DIVISION.                                 |
|FILE SECTION.                                  |
|FD  ENTDATAS    BLOCK 0     RECORDING F        |
|    LABEL RECORDS STANDARD.                    |
|                              +----------------------------+
|                              | IDENTIFICAÇÃO DE AREAS I/O |
|                              +----------------------------+
|                                               |
| FD ENTCADAS                                   | CadastroMutuarios entcadas;
|                                               | CadastroMutuarios.Record entcadas_Record;
| 01 REG-CAD-EN                                 | CadastroMutuarios.Record regCadEn;
| FD ENTDATAS                                   | DatasIndiceBNH entdatas;
|                                               | DatasIndiceBNH.Record    entdatas_Record;
| 01 CREIND1-EN                                 | DatasIndiceBNH.Record_01 creind1En;
| 01 CREIND2-EN                                 | DatasIndiceBNH.Record_02 creind2En;
| FD SAILISTA                                   | JapiReportFile sailista;
|                         +---------------------------------------+
|                         | IDENTIFICAÇÃO DE INTERFACES DE USUÁRIO |
|                         +---------------------------------------+
|                                               |
| 01 CABEC                .                     |
|    02 FILLER         PIC X(112) VALUE         |
|          '  FBNH2010    *** C R E D I T O     |
|-         '  I M O B I L I A R I O             |
|-         '*** L I S T A G E M   D O           |
|-         '   C A D A S T R O'                 |
|    02 FILLER         PIC X(3) VALUE 'FL.'     |
|    02 FOLHA-CAB      PIC ZZ.Z99               |
|    02 FILLER         PIC X(4) VALUE SPACES    |
|    02 DATA-CAB       PIC X(8)                 |
|                                               | StringBuffer cabec = new StringBuffer(
|                                               |     "  FBNH2010    *** C R E D I T O   I M O B      +
|                                               |     "R I O    ***  L I S T A G E M    D O   C A D A  +
|                                               |      " S T R O       FL."                             +
|                                               |     "ZZ.Z99"                                          +
|                                               |     "    "                                            +
|                                               |     "XXXXXXXX"
|                                               |     );
|                                               | final Japi.Mask FOLHA_CAB = new
|                                               |       Japi.Mask("ZZ.Z99");
|                                               | final int       $FOLHA_CAB = 115;
|                                               | final int       DATA_CAB = 8;
|                                               | final int       $DATA_CAB = 125;
| 01 DJDE              PIC X(40) VALUE          |
| '1DJDE JDL=DFAULT,JDE=F163S,END;'             |
|                                               | StringBuffer djde = new StringBuffer(
|                                               |     "1DJDE JDL=DFAULT,JDE=F163S,END;         "
|                                               |     );
| 01 LIN-TOTFAIX          .                     |
```

```
    02 FILLER      PIC X(10) VALUE ' FIRMA  *'
    02 FIR-L       PIC 99
    02 FILLER      PIC X(14) VALUE '*    FAIXA '
    02 FAIXA-LIN   PIC 99999.9999
```

```
StringBuffer linTotfaix = new StringBuffer(
    " FIRMA  * "                                                    +
    "99"                        + // pos=10                        +
    "*    FAIXA    "                                                +
    "99999.9999"
    );
final int       FIR_L = 2;
final int       $FIR_L = 10;
final Japi.Mask FAIXA_LIN = new Japi.Mask("99999.9999");
final int       $FAIXA_LIN = 26;
```

Modelo para a extração de código legado Cobol

```
01 TOTAIS.
    02 NOMES-DOS-TOTAIS PIC X(32)
    02 VALORES  PIC B(20)---.---.---.---.--9,99
```

```
StringBuffer totais = new StringBuffer(
    "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXBBBBBBBBBBBBBBBBBB" +
    "BB---.---.---.---.--9,99"
    );
final int       NOMES_DOS_TOTAIS = 32;
final int       $NOMES_DOS_TOTAIS = 0;
final Japi.Mask VALORES = new Japi.Mask
("BBBBBBBBBBBBBBBBBBBB---.---.---.---.--9,99");
final int       $VALORES = 32;
```

```
                +-----------------------------------------+
                |  IDENTIFICAÇÃO DE ÁREAS SIMPLES DE WORK |
                +-----------------------------------------+
77 AC-REG           PIC 9(6) VALUE ZEROS.
```

```
int acReg = 0;
```

```
77 FOLHAS           PIC 9(5) VALUE ZEROS COMP-3.
```

```
int folhas = 0;
```

```
77 TOTAPAR-GER      PIC 9(7) VALUE ZEROS COMP-3.
```

```
int totaparGer = 0;
```

```
                +-------------------------------------------+
                |  IDENTIFICAÇÃO DE ÁREAS COMPLEXAS DE WORK |
                +-------------------------------------------+
02 DATA-INVE            .
    03 DIA              PIC 99
    03 MES              PIC 99
    03 ANO              PIC 99
```

```
StringBuffer dataInve = new StringBuffer("      ");
final int       DIA_OF_DATA_INVE = 2;
final int       $DIA_OF_DATA_INVE = 0;
final int       MES_OF_DATA_INVE = 2;
final int       $MES_OF_DATA_INVE = 2;
final int       ANO_OF_DATA_INVE = 2;
final int       $ANO_OF_DATA_INVE = 4;
```

```
                +-----------------------------------------+
                |  IDENTIFICAÇÃO DE ARRAYS DE UMA DIMENSÃO |
                +-----------------------------------------+
01 ACUMULADORES         .
 02 FILLER      PIC 9(15)V99 VALUE ZEROS COMP-3
 02 ACTXSEGUR   PIC 9(15)V99 VALUE ZEROS COMP-3
 02 ACTXCOBRA   PIC 9(15)V99 VALUE ZEROS COMP-3
 02 ACJUREMPR   PIC 9(15)V99 VALUE ZEROS COMP-3
 02 ACAMOREM    PIC S9(15)V99 VALUE ZEROS COMP-3
01 ACUM-R      REDEFINES ACUMULADORES
 02 ACUM-TAB    PIC 9(15)V99 COMP-3 OCCURS 5
```

```
BigDecimal actxsegur = BigDecimal.valueOf(0L, 2);
BigDecimal actxcobra = BigDecimal.valueOf(0L, 2);
BigDecimal acjurempr = BigDecimal.valueOf(0L, 2);
BigDecimal acamorem = BigDecimal.valueOf(0L, 2);
BigDecimal [] acumTab = {
    actxsegur,
    actxcobra,
    acjurempr,
    acamorem,
    };
```

```
01 NOMES-TODOS-TOTAIS    .
    02 FILLER          PIC X(32) VALUE
       '  TOTAL DE CONTRATOS EM VIGOR   '
    02 FILLER          PIC X(32) VALUE
       '  TOTAL DE CONTRATOS PARALISADOS'
    02 FILLER          PIC X(32) VALUE
       '  TOT. GERAL DA PRESTACAO       '
```

```
| 01 TOTAIS-1-R    REDEFINES NOMES-TODOS-TOTAIS  |
|    02  TOT-TAB       PIC X(32) OCCURS 3 TIMES  |
```

```
| String[] totTab = {
|     null,
|     "  TOTAL DE CONTRATOS EM VIGOR    ",
|     "  TOTAL DE CONTRATOS PARALISADOS",
|     "  TOT. GERAL DA PRESTACAO        "
|     };
```

Modelo para a extração de código legado Cobol

```
                          +-------------------------------------+
                          |  IDENTIFICAÇÃO DE REGRAS DE NEGÓCIOS |
                          +-------------------------------------+
|PROCEDURE DIVISION                          |Method: null  public int fbnh2010_main() throws Exception {
| Define FINALIZAR                           | boolean finalizar = false;
| Define CARREGA-TOT-FAIXA                   | boolean carregaTotFaixa = false;
| Define QUEBRA                              | boolean quebra = false;
| Define ADICIONAR                           | boolean adicionar = false;
| Define LISTAR                              | boolean listar = false;
| OPEN INPUT ENTCADAS ENTDATAS OUTPUT SAILISTA|
                                             | entcadas = new CadastroMutuarios();
                                             | entcadas.openInput();
                                             | entdatas = new DatasIndiceBNH();
                                             | entdatas.openInput();
                                             | sailista = new JapiReportFile();
                                             | sailista.openOutput("C:/FBNH/REPORT/fbnh2010.rpt", 65);
| WRITE RELATO FROM DJDE[W/40C] AFTER CANAL-1|
                                             | sailista.nextPage();
                                             | sailista.write(djde);
| READ ENTDATAS AT                           | entdatas_Record = entdatas.read();
                                             | if (entdatas_Record != null) {
                                             |  if (entdatas_Record instanceof DatasIndiceBNH.Record_01)
                                             |   creind1En = (DatasIndiceBNH.Record_01) entdatas_Record;
                                             |else if (entdatas_Record instanceof DatasIndiceBNH.Record_02)
                                             | creind2En = (DatasIndiceBNH.Record_02)entdatas_Record;
                                             | }
| END                                        | else {
|  DISPLAY  'TERMINO DE DATAS INDEVIDO'      |
                                             | Japi.display("TERMINO DE DATAS INDEVIDO");
|  WRITE RELATO FROM DJDE[W/40C] AFTER CANAL-1|
                                             | sailista.nextPage();
                                             | sailista.write(djde);
|  CLOSE ENTCADAS ENTDATAS SAILISTA          |
                                             | entcadas.close();
                                             | entdatas.close();
                                             | sailista.close();
|  STOP RUN                                  | return rc;
                                             | }
| MOVE DATMVDMA-EN TO WRK-DT-EDIT            |
                                             | Japi.overlay(wrkDtEdit.length(), +
                                             | creind1En.datmvdma, 0, wrkDtEdit, 0);
| MOVE WRK-DT-DM TO WRK-CAB-DM               |
                                             | Japi.overlay(WRK_CAB_DM, wrkDtEdit, $WRK_DT_DM, +
                                             | wrkDtCab, $WRK_CAB_DM);
| MOVE WRK-DT-AA TO WRK-CAB-AA               |
                                             | Japi.overlay(WRK_CAB_AA, wrkDtEdit, $WRK_DT_AA, +
                                             | wrkDtCab, $WRK_CAB_AA);
| MOVE WRK-DT-CAB TO DATA-CAB                |
                                             | Japi.overlay(DATA_CAB, wrkDtCab, 0, cabec, $DATA_CAB);
| LE-CAD: While(true)                        | leCad: while (true) {
|  READ ENTCADAS AT                          | entcadas_Record = entcadas.read();
                                             | if (entcadas_Record != null)
                                             | regCadEn = entcadas_Record;
|  END                                       | else {
|  Set&Break FINALIZAR                       | finalizar = true;
                                             | break leCad;
                                             | }
|  ADD 1 TO CB-ON-001                        | cbOn001++;
|   IF CB-ON-001 = 1                         | if (cbOn001 == 1) {
|    MOVE FIRMA-CAD-EN TO FIRMA-W            | firmaW = regCadEn.firma;
|     MOVE FAIXA-CAD-EN TO FAIXA-W           | faixaW = regCadEn.faixa;
|     Set CARREGA-TOT-FAIXA                  | carregaTotFaixa = true;
                                             | }
|   ELSE                                     | else {
|   IF  FIRMA-CAD-EN != FIRMA-W              | if (regCadEn.firma != firmaW)
|       Set QUEBRA                           | quebra = true;
                                             | }
|   JAPI-TESTA-INDF: While(true)             | japiTestaIndf: while (true) {
|      Test !QUEBRA & !CARREGA-TOT-FAIXA     | if (!quebra && !carregaTotFaixa) {
```

```
       PERFORM TESTA-INDF                          | testaIndf();
        Comment: VER-QUEBRA-FAIXA                  | // VER QUEBRA FAIXA
       IF FAIXA-CAD-EN = FAIXA-W                   | if (regCadEn.faixa != faixaW) {
        ADD 1 TO FOLHAS                            | folhas++;
       MOVE FOLHAS TO FOLHA-CAB                    | Japi.overlay(FOLHA_CAB.length, +
                                                   | Japi.edit(folhas, FOLHA_CAB), 0, cabec, $FOLHA_CAB);
       WRITE RELATO FROM CABEC AFTER CANAL-1       | sailista.nextPage();
```

Modelo para a extração de código legado Cobol

```
                                                   | sailista.write(cabec);
       WRITE RELATO FROM LINHA-TOT-CODFA AFTER      | sailista.skipLines(2);
                                                   | sailista.write(linhaTotCodfa);
         MOVE FIRMA-W TO FIR-L                     | Japi.overlay(FIR_L, firmaW, 0, linTotfaix, $FIR_L);
         MOVE FAIXA-W TO FAIXA-LIN                 | Japi.overlay(FAIXA_LIN.length, +
                                                   | Japi.edit(faixaW, FAIXA_LIN), 0,linTotfaix, $FAIXA_LIN);
         WRITE RELATO FROM LIN-TOTFAIX AFTER 2     | sailista.skipLines(2);
                                                   | sailista.write(linTotfaix);
         ADD 1 TO INDF                             | indf++;
         PERFORM DAR-TOTAL-FAIXA UNTIL INDF        |
         GREATER 12                                | while (indf <= 12)
                                                   |  darTotalFaixa();
         MOVE ZEROS TO INDF                        | indf = 0;
         MOVE ACUM-TXSEG TO ACTXSEG-L              | Japi.overlay(ACTXSEG_L.length, +
                                                   |Japi.edit(acumTxseg, ACTXSEG_L), 0, linhaFaixa, $ACTXSEG_L);
         MOVE ZEROS TO ACUM-TXSEG                  |
                       ACUM-TXCOB                  |
                       ACUM-JUREMP                 |
                       ACUM-AMOEMP                 |
                                                   | acumTxseg = BigDecimal.valueOf(0L, 2);
                                                   | acumTxcob = BigDecimal.valueOf(0L, 2);
                                                   | acumJuremp = BigDecimal.valueOf(0L, 2);
                                                   | acumAmoemp = BigDecimal.valueOf(0L, 2);
                                                   | }
                                                   |  }
                                                   |
 Reset FINALIZAR                                   | finalizar = false;
 PERFORM  QUEBRA                                   | quebra();
 SOMA-INDF2B: While(true)                          | somaIndf2b: while (true) {
   ADD 1 TO INDF2                                  | indf2++;
   IF INDF2 GREATER 12                             | if (indf2 > 12)
   MOVE 11 TO IND                                  | ind = 11;
 PERFORM LISTA-TOT UNTIL IND GREATER 16            | while (ind <= 16)
                                                   | listaTot();
 MOVE AC-REG1 TO VALORES-1                         | Japi.overlay(VALORES_1.length,+
                                                   | Japi.edit(acReg, VALORES_1), 0, totais1, $VALORES_1);
 WRITE RELATO FROM TOTAIS-1 AFTER 2                | sailista.skipLines(2);
                                                   | sailista.write(totais1);
 CLOSE ENTDATAS ENTCADAS SAILISTA                  | entdatas.close();
                                                   | entcadas.close();
                                                   | sailista.close();
 STOP RUN                                          | return rc;
```

3.6 Conversão – reengenharia de sistemas – migrar o código legado

A engenharia reversa [9] surge como opção para a recuperação de um modelo em nível de abstração mais alto que a do código fonte. Se além desse modelo houver interesse na mudança da linguagem de programação, um processo de reengenharia deve ser aplicado.

Sistema orientado a objetos mesmo não sendo ainda de amplo domínio da comunidade de TI é bem aceito como sendo uma das soluções para os problemas de manutenção e reuso de sistemas, através da utilização de seus conceitos [30]. Sendo assim, os sistemas legados poderiam ser redesenvolvidos ou sofrerem processo de reengenharia orientada a objetos. A primeira opção deve ser descartada devido ao alto custo e também ao longo tempo necessário para que a nova aplicação atinja o mesmo desempenho do antigo sistema [6] .

A reengenharia orientada a objetos então deve ser utilizada para a migrar os códigos legados em projetos de migração. Como a primeira etapa de um processo de reengenharia é a engenharia reversa, uma abordagem de engenharia reversa orientada a objetos torna-se extremamente necessária.

A seguir é apresentado de forma gráfica o modelo que a partir de um programa Cobol, do tipo monolítico (uma camada) possa ser migrado para o modelo em três camadas.

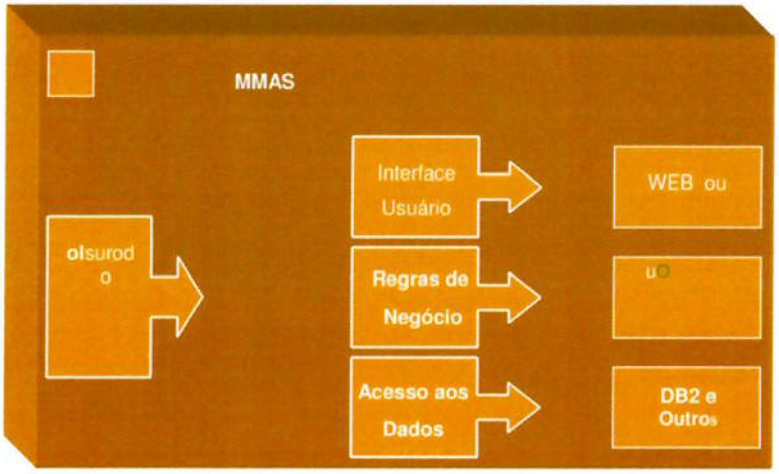


Figura 10 Conversão de Sistemas Legados

3.6.1 Etapas para migrar códigos legados procedimental para código orientado a objeto

O resultado da fase 3.5.1 - primeira fase, é um programa estruturado, arquivado ainda em COBOL em forma de modelo lógico, que servirá de entrada para converter o código de COBOL para a linguagem JAVA nas etapas seguintes. Os procedimentos são os apresentados em todas as atividades no decorrer do item 3.5.

A seguir são descritas as etapas e experiências em ordem cronológica do processo para transformar códigos legados em COBOL em linguagem JAVA em três camadas.

- **Transformar o código procedimental em uma única classe - primeiro esforço**

A principal atividade nesse processo é transformar o código procedimental COBOL em um código intermediário, que servirá de entrada nas etapas seguintes para a geração do código final na tecnologia JAVA. Assumiu-se que era possível dividir, automaticamente, os componentes de um programa estruturado em classes, sem intervenção humana, através de ferramentas que identificam e geram esse código intermediário (isso já possível com ferramentas desenvolvidas por empresas especializadas em re-engenharia de sistemas) . O algoritmo usado foi baseado na montagem e simulação de um modelo para essa finalidade.

- **Extração do conhecimento – separar as classes do código em 3 camadas**

A experiência anterior disponibilizou um código, já numa linguagem intermediária considerando os critérios de decomposição do código original em COBOL em uma classe monolítica. A etapa seguinte será a separação das classes em três classes: camada de apresentação, de dados e de negócios. Foram identificadas heurísticas: acoplamento de dados, chamadas/invocação de programa, e de variáveis. Todas apresentam inconvenientes: acoplamento relativo ao fluxo de controle é inferior ao acoplamento de dados; variáveis poder ser organizadas em estruturas que podem ser consideradas classe (ou simples depósitos temporários). Para obter programas estruturados com organização conforme orientação a objetos, necessárias análises mais profundas das variáveis e sua utilização. Concluiu-se que nem sempre a transparência do programa fica melhor - os resultados devem ser criticamente avaliados e ajustados.

- **Aplicabilidade de ferramentas para reengenharia de sistemas**

Migrar sistemas legados em grandes corporações através de um processo de reengenharia de sistemas sem o auxilio de uma ferramenta que possa interpretar o código legado e efetuar a reengenharia, talvez não fosse impossível, mas certamente, por processo manual, levaria muito tempo e o custo seria proibitivo para essa atividade, sem contar os erros que poderiam advir desse processo. Portanto os estudos acadêmicos, objeto de estudo para essa dissertação limitam-se tão somente às técnicas que devem ser utilizadas para a reengenharia de sistemas legados.

O que se propõe, além das técnicas já conhecidas para a reengenharia de sistemas legados, é a utilização de ferramentas que possam automatizar esse processo de entendimento do código de um legado e fazer, não apenas a reengenharia desse legado, mas também fazer a engenharia reversa, pois legados conhecidos quase não têm documentação das suas regras de negócios, e isso poderia ser um bom motivo também para recuperar essa documentação.

- **Metodologia para geração de código OO**

Uma ferramenta (Se construída para essa finalidade) permite visualizar a representação interna de um programa, que consiste na análise dos códigos fontes originais e a geração de um código OO. Os seguintes produtos serão gerados, a partir dos códigos fontes de telas e programas, os quais serão convertidos para os elementos correspondentes, descritos nos itens seguintes.

3.6.2 Produtos gerados

Os produtos gerados têm por base a arquitetura IBM a partir de uma engenharia reversa do código de interface para a reengenharia dos sistemas legados. Os seguintes produtos serão gerados pelo modelo de migração apresentado:

- **Geração do objeto apresentação a partir de Relatório, de mapa ou telas BMS/ MFS**

Etapa em que todas as API´s, tanto dos Relatórios ou dos mapas do BMS do CICS como os mapas MFS do IMS/DC , serão transformados em objetos JAVA da seguinte forma:

- Relatórios, tal como no processo Batch será formatado, para serem gerados e possibilitar a sua impressão em papel;
- Uma página JSP contendo a parte de apresentação, para a interface com o usuário.
- Um objeto JavaBean que interagirá com a página JSP para controlar o fluxo de dados. O papel deste objeto será o mesmo que o CICS o IMS/DC e o COMS têm com relação às telas, ou seja, será responsável pelo tratamento dos dados de entrada e de saída que constam da página JSP e o controle dos botões que correspondem às PF's em um terminal sem inteligência.
- Um objeto Servlet que interagirá com o programa ou com o seu objeto correspondente convertido e a página JSP, assim como o encadeamento dessas páginas JSP entre si.

➢ **Geração do Objeto programa**

Os produtos gerados têm também por base a arquitetura IBM, a partir de uma engenharia reversa do código para a reengenharia dos sistemas legados. Os seguintes produtos serão gerados pelo modelo de migração apresentado:

- Um objeto EJB, do tipo "Stateless Session Bean", que conterá a lógica principal;
- Objetos que podem ser exportados ou acoplados a ferramentas geradoras ou de engenharia de software, tais como WSAD da IBM ou Rational Rose;
- Os códigos gerados herdam todos os comentários do código original;
- Código gerado otimizado para tirar proveito da tecnologia OO;
- Ênfase na geração de código tendo como preocupação à performance.

➢ **Geração do Objeto acesso aos dados**

Os produtos gerados têm por base a arquitetura IBM ou UNISYS, a partir de uma engenharia reversa dos acessos às bases de dados para a reengenharia dos sistemas legados. Os seguintes produtos serão gerados pelo modelo de migração apresentado:

- Um ou mais objetos para cada arquivo seqüencial utilizado.
- Um ou mais objetos EJB, do tipo "Bean Managed Persistence Entity Bean" para cada tabela DB2 acessada. Em alguns casos, por motivo de performance, o objeto não será um EJB, isto é, será um objeto normal, o qual encapsulará a tabela DB2 de modo similar definido num Entity Bean.
- Se o programa acessar um arquivo VSAM KSDS ou RRDS, este será convertido em uma tabela DB2, e a mesma será acessada por um objeto criado da mesma forma que um arquivo seqüencial, porém, via JDBC.
- No caso de relatórios na modalidade de processamento não On-line (batch), ou seja, as SYSOUTs, serão utilizado, a princípio, um objeto genérico, visto que os relatórios são simples demais para possuírem um objeto exclusivo. No futuro, porém, será estudada a possibilidade de se gerar classes mais elaboradas para tratar a impressão desses relatórios, assim como o encaminhamento para o seu destino.
- Com relação aos relatórios na modalidade on-line, conhecidas também como tele-impressão, terá que ser estudada uma maneira de contemplá-los. Quando se encontrar uma forma de imprimir e encaminhar os relatórios não On-line (Batch) mencionado anteriormente, este também será o meio utilizado nos casos de tele-impressão.

3.6.3 Analisar o código legado

Identificar código no sistema legado, que invoca funcionalidades duplicadas e determinar como modificar esse código para usar novos serviços, bem como avaliar a performance do código gerado. O código gerado deverá ter no mínimo a performance equivalente do sistema legado. O usuário não poderá ser penalizado quanto a esse ponto.

3.7 TESTES DA APLICAÇÃO MIGRADA

Demonstrar que a aplicação migrada atinge os objetivos da migração quando roda no ambiente alvo com os novos serviços, sem diferenças quanto ao conteúdo dos dados tratados, APIs, Relatórios e Data Bases, atendendo plenamente aos requisitos inicias da aplicação legada.

3.7.1 Testes Individuais

Demonstrar individualmente que os programas migrados, AGORA classes, atingem os seus objetivos quando rodam no ambiente alvo com os novos serviços, sem diferenças quanto ao conteúdo de dados tratados, apresentações e relatórios.

3.7.2 Testes Integrados

Demonstrar que o conjunto de classes migrado atinge os seus objetivos quando rodam no ambiente alvo com os novos serviços, sem diferenças quanto ao seu conteúdo de dados tratados, apresentações e relatórios, integrando-se a todo ambiente de testes.

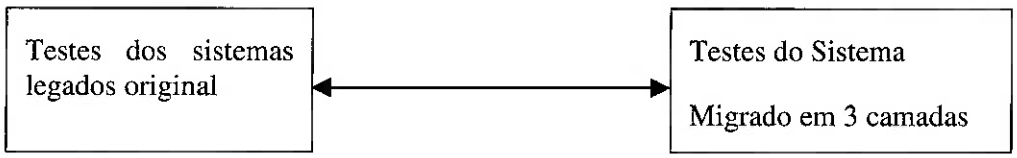


Figura 11 Testes nos dois ambientes e comparação de resultados

3.7.3 Validar e Homologar o sistema migrado - comparando resultados

Num projeto que envolva um grande número de programas, e que envolva mudanças tecnológicas, principalmente as que afetam o usuário, dever-se-á ter um rígido controle sobre a fase de testes do projeto. Para o usuário final, o sistema migrado deve se comportar como se nada estivesse acontecendo, a não ser pelo "layout" das telas dos sistemas On-line, que

provavelmente estará mudando em função do sistema que estará operando com interface gráfica na interação com o usuário assim como em alguns casos pela engenharia do produto que estará operando com orientação a objetos. **COMPARAR todas as saídas** – Data Bases, APIs, relatórios contra os sistemas **ORIGINAIS** em COBOL que serviram de parâmetros para os testes, utilizando dados iguais aos utilizados pelo sistema migrado. Todos os arquivos convencionais envolvidos, os bancos de dados dos sistemas, e as "telas" (API) devem ser **EXATAMENTE IGUAS** em ambos os sistemas para que o usuário possa "homologar" a migração do sistema legado.

3.8 IMPLANTAR EM PRODUÇÃO O SISTEMA LEGADO

Implantar em produção a nova aplicação, já em tecnologia OO. Esta etapa encerra todos os processos de um projeto de reengenharia, de um sistema legado numa corporação. Esta etapa requer um acompanhamento sistemático do pessoal envolvido nos processos, da área de TI, para solucionar eventuais problemas, já que depois de uma aplicação implantada é difícil retornar à aplicação legada, pois muitas atividades ocorreram, principalmente alterações nas bases de dados. Requer também o envolvimento sistemático dos usuários das aplicações transformadas, para de imediato comunicar os problemas que eventualmente possam ocorrer.

CAPÍTULO 4 CONSIDERAÇÕES FINAIS

Durante as três primeiras décadas da era do computador, o principal desafio era desenvolver um hardware que reduzisse o custo de processamento e armazenagem de dados. Ao longo da década de 1980, avanços na microeletrônica resultaram em maior poder de computação a um custo cada vez mais baixo. Hoje o problema é diferente. O principal desafio durante a década de 1990 foi melhorar a qualidade e reduzir o custo de soluções baseadas em computador. Soluções que são implementadas com software. O poder de um computador central (mainframe) da década de 1980 agora está à disposição sobre uma escrivaninha. As assombrosas capacidades de processamento e armazenagem do moderno hardware representam um grande potencial de computação. O software é o mecanismo que nos possibilita aproveitar e dar vazão a esse potencial.

4.1 CONCLUSÕES

O objetivo inicial desse trabalho foi elaborar uma proposta que possibilitasse que as corporações pudessem efetuar reengenharia de seus sistemas legados a partir de linguagens do tipo procedimental, escritos em COBOL, para sistemas orientados a objetos, e apresentados em três camadas.

Verificou-se então que com as estratégias e procedimentos apresentados ao longo desse trabalho que é possível efetuar a reengenharia de sistemas legados, utilizando-se dos modelos apresentados neste trabalho.

No início desse trabalho, avaliou-se em implementar nas estratégias e procedimentos, um modelo que contemplasse além da reengenharia, um mecanismo que durante o processo fosse possível implementar **NOVOS CÓDIGOS** de forma automática, em todos as classes que fossem afetadas por essa necessidade, para atender a novos requisitos dos usuários que eventualmente pudessem estar represadas, e que de certa forma seria de relevante interesse para processos de reengenharia de sistemas legados.

Concluindo, entre tantos benefícios a reengenharia de software vem contribuir para a diminuição dos custos em desenvolvimento de sistemas, melhorar a qualidade dos produtos disponibilizados aos usuários, reduzir tempo de desenvolvimento de sistemas e a tecnologia orientação a objetos é a responsável e quem viabiliza essas mudanças, e para melhor.

O objetivo inicial de elaborar uma "Proposta de uma estratégia de reengenharia de sistemas legados para sistema orientado a objeto" foi atingido.

4.2 COMENTÁRIOS GERAIS

A dificuldade de conciliar as atividades profissionais em função de uma estrutura empresarial, com atividades de especialização motivou atrasos constantes. Espera-se que uma nova estrutura empresarial que está sendo implementada, possa no futuro não mais ser motivos de atrasos em projetos acadêmicos.

4.3 TRABALHOS FUTUROS

A especificidade de reengenharia de sistemas legados, o que o mercado está buscando para melhorar os seus sistemas legados, com a implementação de um modelo **AUTOMATIZADO** que auxilie as organizações a implementar novos códigos durante um processo de reengenharia de sistemas legados, poderá ser objeto de trabalhos futuros.

4.4 CONTRIBUIÇÃO

Este trabalho trouxe contribuições que podem ajudar a reengenharia de sistemas legados uma atividade mais segura e principalmente isentar os processos de erros humanos, apresentado um padrão de reengenharia de legados.

REFERÊNCIAS BIBLIOGRÁFICAS UTILIZADAS

[1] Beck, Kent; Cunningham, Ward. A Laboratory for Teaching Object Oriented Thinking, OOPSLA 89 Proceedings

[2] Becker, Karin; Zanella, Ana L.; Collaborative Design Course to Teach Object Oriented Oriented Thinking, PUCRS, ASOO98, 1998
http://www.inf.pucrs.br/~kbecker/ppubl.html

[3] Bennet K, Legacy System: Coping with sucess "IEEE Software, january, 1995"

[4] Brodie, M. L.; and Stonebraker, M.; "DARWIN: On the Incremetal Migration of Legacy Information Systems," Technical Memeorandum, Eletronics Research Laboratory, College of Engineering, University of California, Berkeley, March 1993.

[5] Brodie, M. L.; and Stonebraker, M.; Migration Legacy Systems: Gateways, Interfaces & the Incremental Approach, Morgan Kauffman, 1995

[6] Camargo, V.V., e Penteado, R. Ap. D., " Diretrizes para Engenharia Reversa Orientada a Objetos de Sistemas COBOL com a utilização do Método Fusion/RE" Universidade federal de São Carlos - UFSCar – Depto de Computação.

[7] Canning, R, "The Maintenance 'Iceberg', " EDP Analyzer, vol. 10, no. 10, October 1972.

[8] Cerdinley, E., abd Daí H., "Encapsulation,- na Issue for legacy Systems" BT technology Journal, july 1993.

[9] Chikofsky, E and Cross, J., "Reverse Engineering and Design Recovery – A Taxonomy" IEEE Software, january 1990.

[10] Dedene, G.; DeVreese, J.; "Realities of Off-Shore Reengineering", IEEE Software,

january 1995, pp. 35-45

[11] Gotlieb, L., "Learning to Live with Legacy Systems" CMA, may 1993.

[12] Hanna, M., "Maintenance Burden Begging for a Remedy," Datamation, April 1993, pp53-63.

[13] Jacobson, Ivar; Fredrick Lindstrom - Re-engineering of old Systems to na Object-oriented Architecture – OOPSLA 91, pp 340-350

[14] John, Bergey; Liam, O'Brien; Dennis, Smith – DoD Software Migration Planning Technical Note CMU/SEI-2001-TN-012

[15] John, Bergey; Dennis, Smith; Nelson Weiderman – DoD Legacy System Migration Guidelines - Technical Note CMU/SEI-2001-TN-013

[16] Mayers, C., "Legacy data Access" ANSA User Group Meeting, Cambridge, England, April 1994.

[17] Nassif, R.; Mitchussen, D.; " Issues and Approaches for Migration/Cohabitation Between Legacy and New Systems", SIGMOD '93, International Conference on Management of Data – May 1993, pp 471-471

[18] Nimer, Fernando. Vale a pena investir em Orientação a Objetos, Developers 'Magazine', n. 12, pp 12-15, agosto 1997.

[19] Osborne, W.M. and E.J. Chikofsky, "Fitting Pieces to the Maintenance Puzzle,", IEEE Software, January 1990, pp.10-11.

[20] Roger S. Pressman - Software Engineering – A Practitioner's Approache Mc Graw Hill

[21] Rosana Teresina Vaccare Braga – "padrões de Software a partir da Engenharia Reversa de Sistemas Legados" – Tese de Dissertação – USP – São carlos Novembro 1998.

[22] Santigo, Comella Dorda; Grace A Lewis; Pat Place; Dan, Plakosh; Robert C. Seacord Incremental Modernization of Legacy Systems – Technical Note CMU/SEI-2001-TN-006

[23] Sneed, H., "Planning the Reengineering of Legacy Systems," IEEE Software, January 1995, pp.24-25.

[24] Schick, K., "The Key to Client/Server – Unlocking the Power legacy Systems "Gartner Group Conference, February 1995.

[25] Swanson,E.B., "The Dimensions of Maintenance,"Proc.Second Intl. Conf.Sftware Engineering,IEEE, October 1976,pp.492-497.

[26] Stephanie Wilkinson " From the Dustbin, Cobol Rises – eWeek, May 28, 2001" 1

[27] William Ulrich – Legacy Systems Transformation Strategies – Ed. Prentice Hall

[28] Umar, Amjad - Application (Re)Engineering – Building Web-Based Applications and Dealing with Legacies – Prentice Hall

[29] Wilkening, D. E.; Loyall, J. P.; Pitarys, M.J. e Littlejohn, K. A - Reuse Approach to Software Reengineering. Journal Systems Software, V.30 1995.

[30] WIGGERTS, T., BOSMA, H., AND FIELT, E. Scenarios for the identification of objects in legacy systems. In 4th Working Conference on Reverse Engineering (1997), IEEE Computer Society, pp. 24-32.

[31] Zanella, Ana Lúcia. Apoio ao Ensino de Projeto de Software Orientado a Objetos Baseado na Metodologia CRC, PRCRS, Plano de Estudo e Pesquisa, janeiro 1997, http://tinos.pucrs.br/~azanella/mestrado/pep.htm

REFERÊNCIAS BIBLIOGRÁFICAS RECOMENDADAS

[32] Coleman, D; Malan R., Letsinger, R., "Object-Oriented Development at Work" Prentice Hall -1995.

[33] Coleman, D; Arnold, P.; Bodoff, S.; Dollin, C.; Gilchrist, H.; Ayes, F.; Jeremaes, P.; Desenvolvimento orientado a objetos: o método Fusion, Rio de Janeiro, Campus,1996.

[34] Feltrim, V. D., e Fortes, R.P.M, e Silva, W. F., " Aspectos de validação do Método de engenharia reversa Fusion-RE/I aplicado a um sistema Hipermídia, ICMC - USP

[35] Georgia Tech's Reverse Engineering Group
http://www.cc.gatech.edu/reverse/

[36] IEEE Computer Society TCSE Committee on Reverse Engineering and Reengineering
http://www.tcse.org/revengr/

[37] James Rumbaugh, Micheal Blagha, William Premerlani,
Frederick Eddy, William Lorensen - Object-Oriented Modeling and Design
Prentice-Hall

[38] Masiero, P.C., e Germano, F.S.R., e Braga, R.T.V.,
"A Confederation of Patterns for Resource management" ICMC – USP

[39] Reverse Engineering and System Renovation
http://adam.wins.uva.nl/~x/reeng/REanno.html

[40] Serge Demeyer, Stéphane Ducasse, Oscar Nierstrasz – Object-Oriented
Reengineering Patterns - 2003 Morgan Kaufmann Publishers

[41] Technical Council on Sotfware Engineering Ferramentas Case
http://www.qucis.queensu.ca/Software-Engineering/vendor.html

[42] Terekhov, Andrey A. , "Automating language Conversion: A case Study"
St. Petersbug State University, LANT-TERCOM – St. Petrsburg, Russia,

[43] Terekhov, Andrey A. ., "Automated extraction of classes from legacy
St. Petersbug State University, LANT-TERCOM – St. Petrsburg, Russia